Historisierung von Tabellen in Postgis

Warum Historisierung?

Häufig ist es gewünscht, alte Datenstände in einer Datenbank zu rekonstruieren. Zudem ist eine Nachvollziehbarkeit nützlich: wer hat wann welche Werte in welchem Record geändert. Als Basis für Fragestellungen dieser Art kann man eine Historisierungsstrategie anwenden. Ähnliche Ansätze werden seit einiger Zeit bereits im Kanton Solothurn angewendet. Dort werden Daten beim Importieren (von Interlis oder shape) abgeglichen und historisiert. Der hier beschrieben Ansatz geht auf eine Präsentation der Firma Varlena zurück. (Quelle: http://www.varlena.com/GeneralBits/Tidbits/tt.pdf). In früheren PostgreSQL Versionen (bis Version 6) war „TimeTravel“ fixer Bestandteil der Datenbank. Dieses Feature wurde jedoch in späteren Versionen entfernt, da der überwiegende Anteil der Nutzer eher an besserer Performanz als an Historisierung interessiert war.

Bei dieser Strategie werden Records nie gelöscht und Records bei Änderungen nicht überschrieben. Es werden 2 Datumsstempel verwendet: „create_date“, wird beim Anlegen eines Records auf den aktuellen Zeitstempel gesetzt, und „archive_date“. Das „archive_date“ wird dann gesetzt, wenn der Record durch Löschung oder Aktualisierung obsolet wird. Die jeweils aktuelle Version der Tabelle wird über einen View dargestellt, der diejenigen Records selektiert, die noch kein „archive_date“ gesetzt haben (IS NULL). Ältere Stände können ebenfalls über Views oder SELECT Statements abgeleitet werden in dem über „create_date“ und „archive_date“ die zum damaligen Zeitpunkt gültigen Records selektiert werden. Zusätzlich zur Datumsspalte kann auch der verantwortliche Nutzer/Bearbeiter mitabgespeichert werden („create_user“ und „last_user“).

Nachteile des Ansatzes:
Die Tabelle wächst mit diesem Ansatz beständig, da Daten nie gelöscht oder überschrieben werden. Insbesondere werden die gleichen Geometrien mehrmals abgespeichert, auch wenn nur ein Attributwert im Record sich änderte. Diesen Nachteil könnte man eventuell lösen, indem man die Geometrien in separaten Tabellen ablegt und referenziert. Falls gewünscht, kann man das Archiv auch in einer separaten Tabelle ablegen, anstelle eines Views.

Ein weiterer Nachteil ist, dass manche Datenbankclients Views prinzipiell nicht verändern können (oder wollen). Diese Clients müssen dann mit zusätzlichen „WHERE“ Klauseln im SELECT Statement arbeiten.

Schliesslich ist es nicht ohne Aufwand möglich, Records effektiv zu ändern oder zu löschen (etwa wenn man weiss, dass die Daten falsch erfasst wurden). Um dies zu bewerkstelligen, müssen die Regeln und Trigger temporär deaktiviert werden.

Vorteile des Ansatzes:
Alte Stände können rekonstruiert werden. Die Nutzung ist für den Benutzer sehr einfach. Der View verhält sich wie eine gewöhnliche Tabelle. Die Komplexität geschieht auf Datenbankebene auf dem Server. Die Clientsoftware bekommt davon nichts mit und muss nicht umgeschrieben werden. Allenfalls muss eine Benutzerschnittstelle zur komfortablen Selektion alter Datenstände entwickelt werden.

Die technische Umsetzung

1. Anlegen von zusätzlichen Spalten:

Jeder zu historisierenden Master-Tabelle müssen 5 zusätzliche Spalten hinzugefügt werden: „create_date“ (Typ timestamp), „archive_date“ (Typ timestamp), „create_user“ (Typ text oder Enumeration), „last_user“ (Typ Text oder Enumeration), id (Typ integer, muss gleich wie gid sein). Die id dient zum Verbinden von abgeleiteten Records mit der ursprünglichen gid. Alle abgeleiteten (historisierten) Records haben die gleiche id wie die ursprüngliche gid bei der Erstellung eines Records. Die gid selber muss eindeutig sein, da sie den Primärschlüssel darstellt.

2. Erstellen einer „Delete-Rule“ für die Master-Tabelle

Die Delete-Rule überschreibt das DELETE statement und wandelt es in ein UPDATE Statement um, welches das „archive_date“ auf den jetzigen Zeitpunkt setzt und den „last_user“ auf den aktuellen Datenbank-User setzt.

3. Erstellen einer Triggerfunktion „insert_timegis“

Diese Triggerfunktion wird vor einem neuen INSERT Statement ausgeführt. Sie setzt das „create_date“ auf „now()“ (jetzt), das „archive_date“ auf NULL, die „id“ auf den Wert der Spalte „gid“ und den „create_user“ auf den derzeit eingeloggten Datenbankuser. Diese Triggerfunktion kann in PL/pgSQL geschrieben werden.

4. Erstellen einer Triggerfunktion „update_timegis“

Diese Triggerfunktion führt das UPDATE auf dem aktuellen Record aus, führt aber parallel dazu ein INSERT Statement aus mit einem Record der die alten Daten vor dem UPDATE enthält. Zusätzlich wird in dem neu einzufügenden Record (der die alten Daten enthält), das „archive_date“ auf „now()“ gesetzt, also der Record damit auf obsolet gesetzt. Der Wert des Attributs „id“ wird auf den Wert des ursprünglichen Original-Records gesetzt. Beim aktualisierten Record wird das „create_date“ auf „now“ gesetzt und der „last_user“ auf den „current_user“ gesetzt. Damit die Triggerfunktion generisch für alle GIS-Tabellen funktioniert, wird die Sprache Pl/Perl verwendet. Generische Triggerfunktionen sind derzeit mit Pl/PgSQL nur sehr beschränkt implementierbar.

5. Erstellen von Triggern

Für die Mastertabelle müssen noch 2 Trigger angelegt werden. Diese rufen die beiden generischen Triggerfunktionen (insert_timegis und update_timegis) auf. Es handelt sich um einen INSERT und einen UPDATE Trigger, in beiden Fällen um einen row-level before Trigger.

6. Erstellen eines Views für den aktuellen Datenstand

Schliesslich braucht es einen View (einen Tabellenausschnitt), der aus der Mastertabelle die Records filtert die derzeit aktuell sind. Dies geschieht über einen View, der alle Records selektiert deren archive_date „NULL“ ist.

7. Erstellen eines Views für historische Datenstände (optional)

Um einen alten Stand zu selektieren kann man entweder einen Sub-Selekt über eine WHERE Klausel machen oder einen weiteren View erstellen. Es müssen alle Records selektiert werden, deren „create_date“ älter als der Zeitstempel des gewünschten Datenstands ist und deren

„archive_date“ entweder „NULL“ oder jünger als der Zeitstempel des anvisierten Datenstands ist.

Beispiel :

1. Anlegen von zusätzlichen Spalten:

```
CREATE TABLE test.landwirtschaftsflaechen
(
  gid serial NOT NULL,
  id integer,
  fruchtart text NOT NULL,
  landwirt text NOT NULL,
  area numeric,
  the_geom geometry,
  create_date timestamp without time zone,
  archive_date timestamp without time zone,
  last_user text,
  create_user text,
  CONSTRAINT pkey_lwflaechen_gid PRIMARY KEY (gid),
  CONSTRAINT enforce_dims_the_geom CHECK (ndims(the_geom) = 2),
  CONSTRAINT enforce_geotype_the_geom CHECK (geometrytype(the_geom) =
'MULTIPOLYGON'::text OR the_geom IS NULL),
  CONSTRAINT enforce_srid_the_geom CHECK (srid(the_geom) = 21781)
)
WITH (OIDS=FALSE);
```

Im Beispiel wird eine Tabelle für die Erfassung von Landwirtschaftsflächen angelegt. Neben den normalen Spalten „gid“,“fruchtart“,“landwirt“,“area“ und „the_geom“ werden auch die zusätzlichen Spalten „id“, „create_date“, „archive_date“, „create_user“ und „last_user“ angelegt. Diese 5 Spalten dienen für die Verbindung zur ursprünglichen „gid“, für die zeitliche Einordnung und für die Rückverfolgbarkeit, wer, wann Änderungen durchgeführt hat. Zusätzlich empfiehlt es sich natürlich die entsprechenden Indizes und räumlichen Indizes anzulegen.

2. Erstellen einer „Delete-Rule“ für die Master-Tabelle

```
CREATE OR REPLACE RULE landwirtschaftsflaechen_del AS
  ON DELETE TO test.landwirtschaftsflaechen
  DO INSTEAD UPDATE test.landwirtschaftsflaechen
    SET archive_date = now(), last_user = current_user
   WHERE landwirtschaftsflaechen.gid = old.gid
   AND landwirtschaftsflaechen.archive_date IS NULL;
```

Bei dieser DELETE RULE wird anstelle eines DELETE Statements ein UPDATE Statement ausgeführt und der Record mittels des „archive_date“s auf obsolet gesetzt. Um ein Löschen bereits obsoleter Records zu verhindern wird das UPDATE statement nur durchgeführt wenn das „archive_date“ noch nicht gesetzt ist.

3. Erstellen einer Triggerfunktion „insert_timegis“

```
CREATE OR REPLACE FUNCTION test.insert_timegis()
 RETURNS "trigger" AS
$BODY$
DECLARE
  nr_found_recs integer;
BEGIN
  IF NEW.create_date IS NULL THEN
    NEW.create_date := now();
    NEW.archive_date := NULL;
    NEW.id := NEW.gid;
    -- Remember who created the record
    NEW.create_user := current_user;
    --see if we need to update area
    EXECUTE 'SELECT count(*) FROM information_schema.columns WHERE table_schema = '
     || quote_literal(TG_TABLE_SCHEMA) || ' AND table_name = ' ||
     quote_literal(TG_TABLE_NAME) || ' AND column_name = ''area'';'
    INTO STRICT nr_found_recs;
    IF nr_found_recs = 1 THEN
      NEW.area := ST_AREA(NEW.the_geom);
    END IF;
  END IF;
  RETURN NEW;
END;
$BODY$
LANGUAGE 'plpgsql' VOLATILE;
```

Zusätzlich zum Setzen des Datums und des setzt diese Triggerfunktion auch noch die Fläche einer neu erstellten Geometrie, wenn das Feld „area“ vorhanden ist.

4. Erstellen einer Triggerfunktion „update_timegis“

```
CREATE OR REPLACE FUNCTION test.update_timegis()
  RETURNS "trigger" AS
$BODY$
my ($sql_attrib,$sql_insert,$sql_values,$rv);
if ($_TD->{old}{archive_date} != undef) {
  return "SKIP"; #we don't allow deletion of obsolete records
}
else {
  if ($_TD->{new}{archive_date} == undef) {
    $sql_insert = "INSERT INTO ".$_TD->{table_schema}."."$_TD>{table_name}." (";
    $sql_values = '';
    #prepare SQL statement to get all column_names and data types (udt_name)
    $sql_attrib = "SELECT column_name, udt_name FROM information_schema.columns WHERE
table_schema = '".$_TD->{table_schema}."' AND table_name = '".$_TD->{table_name}."';";
    my $rv = spi_exec_query($sql_attrib);
    my $nrows = $rv->{processed};
    my $areaExists = false;
```

```
    #loop over all column names and concatenate SQL INSERT statement
    foreach my $rn (0 .. $nrows - 1) {
      my $row = $rv->{rows}[$rn];
      if ($row->{column_name} ne "gid") {
        if ($row->{column_name} eq "id") {
          $sql_insert .= "id,";
          $sql_values .= $_TD->{old}{gid}.",";
        }
        elsif ($row->{column_name} eq "archive_date") {
          $sql_insert .= "archive_date,";
          $sql_values .= "now(),";
        }
        else {
          $sql_insert .= "\"".$row->{column_name}."\",";
          if ($_TD->{old}{$row->{column_name}} eq undefined || $_TD->{old}{$row-
>{column_name}} eq "") {
            $sql_values .= "NULL,";
          }
          else {
            if ($row->{udt_name} eq "text" || $row->{udt_name} eq "geometry" || $row-
>{udt_name} eq "timestamp" ) {
              $sql_values .= "'".$_TD->{old}{$row->{column_name}}."',";
            }
            else {
              $sql_values .= $_TD->{old}{$row->{column_name}}.",";
              if ($row->{column_name} eq "area") {
                $areaExists = true;
              }
            }
          }
        }
      }
    }
    chop($sql_insert); #get rid of the last comma
    chop($sql_values); #get rid of the last comma
    $sql_insert .= ") VALUES (".$sql_values.");";
    $rv = spi_exec_query($sql_insert);
    $_TD->{new}{create_date} = "now()";
    # Remember who changed the record
    $rv = spi_exec_query("SELECT current_user;");
    $_TD->{new}{last_user} = $rv->{rows}[0]->{current_user};
    # see if we need to update the area (if existing)
    if ($areaExists) {
      if ($_TD->{new}{the_geom} != $_TD->{old}{the_geom}) {
        $rv = spi_exec_query("SELECT ST_AREA('$_TD->{new}{the_geom}') AS polyarea");
        $_TD->{new}{area} = $rv->{rows}[0]->{polyarea};;
      }
    }
  }
```

```
return "MODIFY";
}
$BODY$
LANGUAGE 'plperl' VOLATILE;
```

Diese Funktion ist in Pl/Perl geschrieben und sollt generisch für alle GIS-Tabellen funktionieren bei denen die Geometrien in der Spalte „the_geom“ abgelegt sind und allfällige Flächenwerte unter „area“. Der aktuelle Layer wird upgedated und mit einem neuen „create_date“ und „last_user“ versehen. Falls die Spalte „area“ existiert und die neue Fläche sich von der alten unterscheidet, wird die Fläche neu berechnet. Für den alten Stand wird ein neuer Record, der die alten Werte enthält und dessen „archive_date“ auf „now()“ gesetzt wird.

5. Erstellen von Triggern

```
CREATE TRIGGER insert_landwirtschaftsflaechen
  BEFORE INSERT
  ON test.landwirtschaftsflaechen
    FOR EACH ROW
    EXECUTE PROCEDURE test.insert_timegis();


CREATE TRIGGER update_landwirtschaftsflaechen
  BEFORE UPDATE
  ON test.landwirtschaftsflaechen
    FOR EACH ROW
    EXECUTE PROCEDURE test.update_timegis();
```

Diese beiden Trigger werden bei INSERT respektive UPDATE Statements ausgeführt. Sie rufen die entsprechenden Triggerfunktionen auf. Beide werden bei jedem sich ändernden Record ausgeführt, und zwar bevor die Daten eingefügt werden.

6. Erstellen eines Views für den aktuellen Datenstand

Der folgende View stellt ein Extrakt der Master-Tabelle mit den derzeit aktuellen Records dar. Die erstellten Views sollten in der Tabelle „public.geometry_columns“ eingetragen werden und der Eintrag nach dem Löschen wieder entfernt werden.

```
CREATE OR REPLACE VIEW test.lwflaechen AS
  SELECT * FROM test.landwirtschaftsflaechen
    WHERE landwirtschaftsflaechen.archive_date IS NULL;
```

Die folgenden RULEs ermöglichen das Editieren des Views:

```
CREATE OR REPLACE RULE "_DELETE" AS
  ON DELETE TO test.lwflaechen DO INSTEAD  DELETE FROM test.landwirtschaftsflaechen
   WHERE landwirtschaftsflaechen.gid = old.gid;


CREATE OR REPLACE RULE "_INSERT" AS
  ON INSERT TO test.lwflaechen DO INSTEAD  INSERT INTO test.landwirtschaftsflaechen
(fruchtart, landwirt, the_geom) VALUES (new.fruchtart, new.landwirt, new.the_geom);
```

```
CREATE OR REPLACE RULE "_UPDATE" AS
  ON UPDATE TO test.lwflaechen DO INSTEAD  UPDATE test.landwirtschaftsflaechen SET
landwirt = new.landwirt, fruchtart = new.fruchtart, the_geom = new.the_geom
    WHERE landwirtschaftsflaechen.gid = new.gid;
```

7. Erstellen eines Views für historische Datenstände (optional)

Für die erstellten Views sollte ein Eintrag für die Geometriespalten in der Tabelle „public.geometry_columns“ vorgenommen werden und der Eintrag nach dem Löschen des Views wieder entfernt werden.

```
CREATE OR REPLACE VIEW test."lwflaechen_2008-05-15" AS
 SELECT * FROM test.landwirtschaftsflaechen
   WHERE landwirtschaftsflaechen.create_date < '2008-05-16 00:00:00'::timestamp
    AND (landwirtschaftsflaechen.archive_date IS NULL OR
landwirtschaftsflaechen.archive_date > '2008-05-16 00:00:00'::timestamp )
   ORDER BY landwirtschaftsflaechen.gid;
```