kanton glarus

Bau und Umwelt
Raumentwicklung und Geoinformation
Kirchstrasse 2
8750 Glarus

WHITEPAPER

Thema **Generische Umsetzung der minimalen Geodatenmodelle in der kantonalen Geodaten-Infrastruktur**

Autor Fachstelle Geoinformation, Dr. Peter Staub

Publikation **2016-02-09**

Änderungshistorie

2016-02-09	Typos; ili2gpkg ergänzt; Abbildungen im Anhang aktualisiert
2016-02-04	komplette Überarbeitung und Erweiterung auf der Basis von ili2pg 3
2015-05-20	aktualisiertes Modellbeispiel mit LV95
2015-04-22	1. Version

Vorwort

Dieses Dokument wurde aus der Motivation heraus verfasst, die Umsetzung des GeoIG einen konkreten Schritt weiterzubringen. Vielerorts werden Konzepte entwickelt, wie konzeptionelle Datenmodelle in produktiven Datenbanksystemen umgesetzt werden können/sollen. Es entsteht der Eindruck, dass dies eine (zu) grosse Herausforderung darstellt und man versuchen muss, auf Seiten der objektorientierten Datenmodellierung die Möglichkeiten einzuschränken, damit die Umsetzung in relationalen Datenbanksystemen verträglicher wird.

Ich bin der Auffassung, dass dies der falsche Ansatz ist. Heute haben wir generische Schnittstellenprogramme zur Hand, welche beliebige Datenmodelle in relationale Datenbankschemata übersetzen und auch zugehörige Transferdaten auf Knopfdruck importieren beziehungsweise exportieren lassen.

Für die vorliegende Publikation konnte ich sehr stark von der Zusammenarbeit mit Kantonsgeometer Stefan Ziegler, Amt für Geoinformation Kanton Solothurn, profitieren. Dafür sei ihm ein grosser Dank gewiss! Im Zuge der Weiterentwicklung des eingesetzten Schnittstellenprogramms ili2pg konnten wir uns intensiv mit Claude Eisenhut, Eisenhut Informatik AG, austauschen. Auch ihm gebührt ein grosser Dank.

★

Der kantonale Raumdatenpool

Die zentrale Datenhaltung in der kantonalen Geodaten-Infrastruktur wird als «Raumdatenpool» bezeichnet. Dabei kommen PostGIS-Datenbanken zum Einsatz, wobei wie üblich zwischen Test/Entwicklung/Integration und Produktion/Publikation unterschieden wird. Die Datennutzung erfolgt neben dem direkten Datenbankzugriff über Kartendienste (Datenviewer oder OGC WMS), Datendienste (OGC WFS) sowie als Datenbezug (Datei-Download).

Gängige Praxis bei der Datenintegration ist häufig immer noch die pragmatische 1:1-Überführung vorhandener, dateibasierter Datenbestände in die Datenbank, um die weitere Datenbearbeitung zu etablieren. Regelmässige Datenlieferungen aus externen Produktionssystemen erfolgen dateibasiert, aber häufig nicht modellbasiert. Im Zuge der Umsetzung des Einführungsgesetzes zum Geoinformationsgesetz (EG GeoIG) im Kanton Glarus müssen Geodatenmodelle entwickelt, adaptiert und nach den Regeln der Kunst umgesetzt werden. Dazu gehören die Datenbankkonfiguration aus konzeptionellen Datenmodellen, der Datenumbau aus den Produktivdaten in die Modellstruktur sowie der Datenexport und die Prüfung gegenüber dem Datenmodell.

Herausforderungen

Methodisches

Grundsätzlich ist es aus methodischer Sicht unerheblich, ob minimale Geodatenmodelle des Bundesrechts (MGDM) oder solche des kantonalen oder kommunalen Rechts (kMGDM) umzusetzen sind. Es wurde an anderer Stelle festgehalten, dass bei Geobasisdaten des Bundesrechts immer ausgehend vom MGDM als Basis etwaige kantonale oder kommunale Mehranforderungen in Form von Modellerweiterungen zu modellieren sind (Abb. 1).

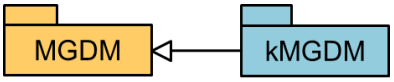

Abb. 1: Kantonale/kommunale Modelle als Erweiterung von Bundesmodellen

Bei der Realisierung wird schnell klar, dass die manuelle Umsetzung des umfangreichen Katalogs minimaler Geodatenmodelle nicht zielführend sein kann. Die Definition der nötigen Datenbankschemata ist aufwändig und kann trotzdem nicht überall eingesetzt werden, weil jede Geodaten-Infrastruktur ihre spezifischen Eigenheiten besitzt oder Konventionen unterworfen ist. Vielmehr muss es das Ziel sein, möglichst generische Schnittstellenwerkzeuge «W» zu erhalten, welche die Modellumsetzung in der Datenbank erledigen (Abb. 2/unten).

Abb. 2: Modellumsetzung mittels Schnittstellenwerkzeug «W»

Bei jedem Datenthema ist grundsätzlich zu entscheiden, ob die Datenproduktion auf die Modellstruktur umgestellt werden soll (Abb. 3, «A») oder ob weiterhin in der bestehenden Datenbankstruktur gearbeitet wird und die modellkonformen Daten bei Bedarf hergeleitet werden (Abb. 3, «B»).

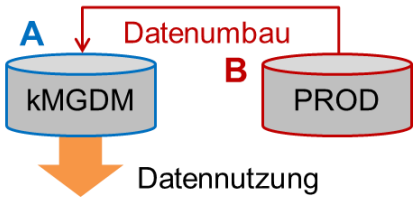

Abb. 3: Produktivdaten im Verhältnis zur Modellstruktur

Die Umstellung ist dann sinnvoll, wenn folgende Punkte mehrheitlich zutreffen:

- das bestehende Datenschema ist wenig umfangreich
- die bestehende Datenstruktur entspricht nicht mehr den Anforderungen
- es handelt sich um ein neues Thema, zu dem bislang noch keine Daten in der Datenbank bearbeitet und gepflegt wurden
- es hängen keine (komplexen) Anwendungen von diesen Daten ab.

Umsetzung objektorientierter Datenmodelle in relationalen Datenbanken

Es ist durchaus nachvollziehbar, dass die Umsetzung eher komplexerer Modellkonstrukte wie beispielsweise Vererbung, Strukturattribute oder etwa die Verwendung von CHBase-Katalogen in relationalen Datenbanken nicht offensichtlich ist. Aus technischer Sicht eröffnen sich zwei Möglichkeiten: entweder, man versucht mit umfangreichen SQL-Skripten die komplette Modellstruktur in der Datenbank abzubilden, oder man vertraut auf die Unterstützung geeigneter, generischer Schnittstellenwerkzeuge, um Datenmodelle umzusetzen.

Die erste Variante scheint zunächst bestechend, weil der Prozess der Abbildung objektorientierter Modellmerkmale in die relationale Datenbank transparent ist und jeder Schritt bewusst kontrolliert werden kann. Andererseits bedeutet dieses Vorgehen aber viel manuellen Konfigurationsaufwand bei gleichzeitiger hoher Fehleranfälligkeit, was zu Inkonsistenzen über mehrere Datenmodelle hinweg führen kann. Die zweite Variante kompensiert diese Nachteile, weil die implementierten Abbildungsregeln bei jeder Anwendung exakt gleich ablaufen. Ein Datenmodell wird in Sekundenschnelle in die Datenbank überführt. Selbstverständlich muss dann vorausgesetzt werden können, dass das verwendete Schnittstellenwerkzeug alle Modellelemente gemäss Spezifikation umsetzen kann. Damit keine intransparenten Blackbox-Lösungen (s. Abb. 2) entstehen, sind OpenSource-Implementierungen mit Offenlegung des Quellcodes zu bevorzugen.

Überführung der Produktivdaten in die Modellstruktur

Eine weitere Hürde ist die Überführung der Produktivdaten in die Modellstruktur. Die historisch gewachsenen Datenschemata der Produktivdaten genügen den konzeptionellen Vorgaben der Datenmodelle nicht immer. Es ist also fallweise zu bestimmen, wie fehlende Modellteile aus dem Datenbestand herzuleiten oder eventuell neu zu erheben sind. Für den Datenumbau kommt das auf geografische Daten erweiterte Konzept *Extract–Transform–Load* (ETL) aus der Businesslogik zur Anwendung. Am «schönsten» ist der Datenumbau mittels SQL-Skripten komplett innerhalb der Datenbank zu erledigen. Dieser Weg hat natürlich den Anspruch, dass über ausreichendes SQL-Wissen verfügt wird. Dafür hat man den Vorteil, dass alle nötigen Prozesse hoch performant und vollständig automatisierbar sind und ohne zusätzliche Software auskommen. Ein äusserst mächtiges und weit verbreitetes kommerzielles Produkt für Geodaten ist die *Feature Manipulation Engine* (FME) der kanadischen Firma Safe Software Inc. Damit können beliebig komplexe Datenumbauprozesse konfiguriert und ausgeführt werden. Mit der Server-Variante von FME können Prozesse einfach automatisiert werden.

Vererbung

Vererbung ist ein in der relationalen Welt vollkommen unbekanntes Konstrukt. Mit der Vererbung wird eine Objekthierarchie eingeführt. Allgemeine Eigenschaften werden in einer – typi-

scherweise abstrakten – Basis- oder Mutter-Klasse definiert und an konkrete Kind-Klassen vererbt. Diese erhalten jeweils alle allgemeinen Eigenschaften aus der Mutter-Klasse, wozu auch Beziehungen gehören können. Damit vermeidet man die redundante Definition gleicher Objekteigenschaften in mehreren Klassen. Bei der Umsetzung solcher Modellkonstrukte ist zu entscheiden, nach welcher Strategie dies erfolgen soll:

- *NewClass-Strategie*: alle Modellklassen, auch die (abstrakte) Mutter-Klasse, werden mit ihren Eigenschaften in Tabellen abgebildet.
- *SubClass-Strategie*: in der Datenbank werden nur die (konkreten) Kind-Klassen, die jeweils alle die Basis-Eigenschaften erhalten, abgebildet.
- *SuperClass-Strategie*: alle Eigenschaften der Kind-Klassen werden in die Mutter-Klasse geschrieben, welche dadurch eigentlich nicht mehr abstrakt ist.

→ Das generische Schnittstellenprogramm soll selbstständig entscheiden können, welche Variante situativ richtig beziehungsweise optimal umzusetzen ist.

Strukturierte Eigenschaften

Strukturen ermöglichen die Definition mehrwertiger Objekteigenschaften. Beispielsweise kann eine Adressen-Struktur definiert werden, die eine Strasse, eine Hausnummer, eine Postleitzahl sowie eine Ortsbezeichnung enthält. Linien- und Flächengeometrien sind eigentlich auch Strukturen: Die Geometrie enthält eine Linie, welche wiederum mehrere Stützpunkte enthält, die schliesslich je ein Koordinatenpaar (oder Tripel) enthalten.

→ Das generische Schnittstellenprogramm soll den Umgang mit Strukturen so einfach wie möglich gestalten: Geometrien, auch «Multi-Geometrien» stehen als Basistypen zur Verfügung; modellierte Strukturattribute werden als separate Tabelle mit entsprechenden Fremdschlüsseln abgebildet. Damit können alle nötigen Informationen geordnet erfasst werden.

Assoziationen

Schliesslich bedarf die Umsetzung von Assoziationen der vertieften Betrachtung. Grundsätzlich können hierarchische Beziehungen mit der Einführung von Fremdschlüsselattributen verhältnismässig einfach realisiert werden (Abb. 4, «`role_A`»). Bei multiplen Beziehungen müssen Hilfstabellen zwischengeschaltet werden, die entsprechend zwei Fremdschlüssel enthalten.

Falls man in der Datenbank den vollständigen Gehalt von Assoziationen nutzen will – insbesondere dann, wenn die Datenproduktion auf die Modellstruktur umgestellt wird – muss das Fremdschlüsselattribut mittels Konsistenzbedingung als «Fremdschlüssel» deklariert werden. Die Beziehungsstärken «Assoziation», «Aggregation» und «Komposition» haben Einfluss auf das Objektverhalten bei Erzeugung, Änderung oder Löschung assoziierter Objekte.

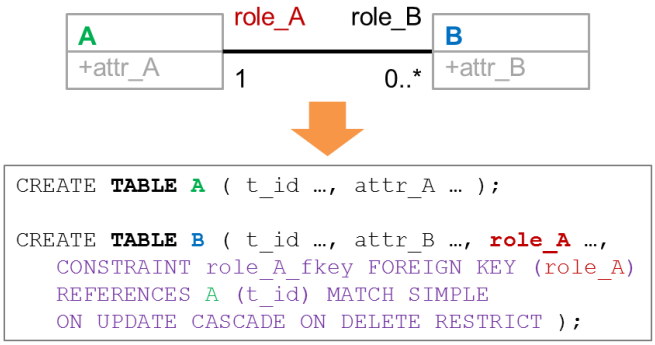

Abb. 4: Beziehung im Modell und auf der Datenbank

→ Das generische Schnittstellenprogramm muss die Bildung von Fremdschlüssel-Konsistenzbedingungen unterstützen und bei multiplen Beziehungen die benötigten Zwischentabellen anlegen.

Das Schnittstellenwerkzeug ili2pg

Technologie

Das Schnittstellenwerkzeug *ili2pg* ist ein offenes Java-Programm, das von der Eisenhut Informatik AG entwickelt wird. Das Werkzeug verfügt über eine einfache grafische Benutzerschnittstelle, die aber nicht den vollständigen Funktionsumfang anbietet. Für die vollumfängliche Nutzung steht ein Kommandozeilenprogramm zur Verfügung. ili2pg ist systemneutral einsetzbar und im Stapelverarbeitungsmodus automatisierbar.

Funktionsweise

ili2pg besitzt drei Hauptfunktionalitäten:

- ein konzeptionelles INTERLIS-Datenmodell in der PostGIS-Datenbank anlegen
- INTERLIS-XML-Transferdaten in die PostGIS-Datenbank laden
- INTERLIS-XML- oder -GML-Transferdaten aus der PostGIS-Datenbank exportieren.

Für alle Funktionalitäten steht eine umfangreiche Reihe von Kommandozeilenoptionen zur Verfügung. So kann beispielsweise ein SQL-Skript erzeugt werden, das die Schema- und Tabellendefinition gemäss Modell enthält. Die Funktionalitäten sind sowohl für INTERLIS 1 als auch für INTERLIS 2.3 nutzbar.

Anforderungen

Im Kontext der Umsetzung von minimalen Geodatenmodellen muss das Schnittstellenwerkzeug ili2pg insbesondere folgende Modellkonstrukte unterstützen: Objektidentifikatoren, Wertebereichsdefinitionen, Strukturen und Strukturattribute, Assoziationen, Referenzattribute, Vererbung, den Umgang mit Datenmodellablagen und mit den CHBase-Modulen.

Umsetzungsprozess

Die Umsetzung der minimalen Geodatenmodelle im kantonalen Raumdatenpool ist als Prozess in fünf Schritten definiert (Abb. 5):

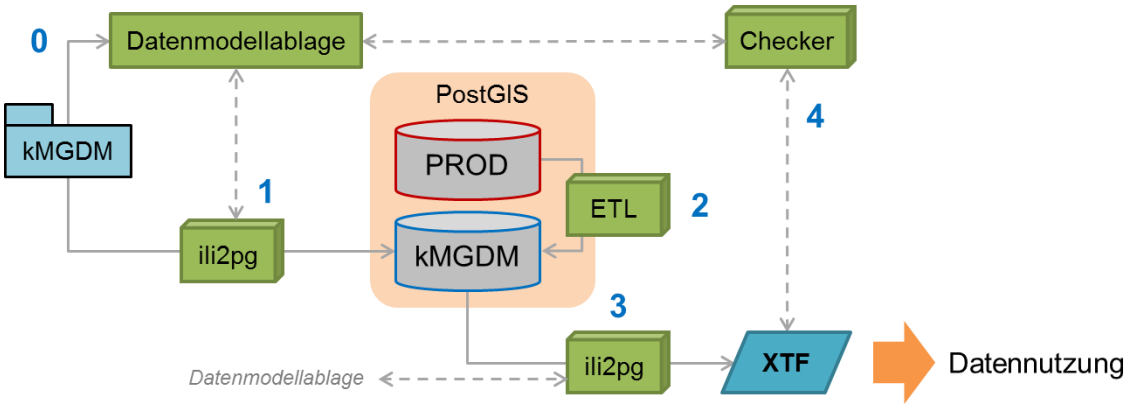

Abb. 5: Umsetzungsprozess

Schritt 0 – Modelldefinition

Die Definition der minimalen Geodatenmodelle ist an zahlreichen anderen Stellen ausführlich beschrieben und wird an dieser Stelle lediglich der konzeptionellen Vollständigkeit halber aufgeführt. Wesentlich ist die Publikation der konzeptionellen Datenmodelle in der Datenmodellablage («Model Repository») des Bundes beziehungsweise des Kantons (kMGDM). Dadurch können die Datenmodelle von den eingesetzten Schnittstellen-, Datenumbau- und Prüfwerkzeugen genutzt werden und müssen nicht zwingend lokal gespeichert werden.

In Abb. 6 ist ein abstraktes Datenmodellbeispiel dargestellt, das in den nachfolgenden Erläuterungen zur Illustration dient.

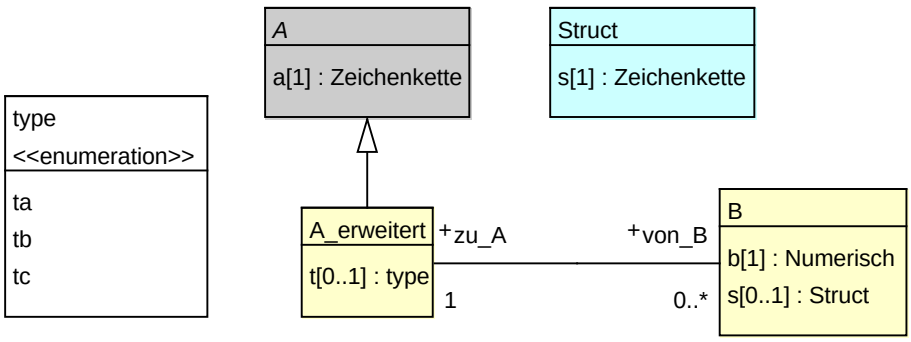

Abb. 6: Fiktives Datenmodellbeispiel

Schritt 1 – Datenmodell in der PostGIS-Datenbank anlegen

Mit dem Schnittstellenwerkzeug ili2pg wird die INTERLIS 2.3-Modelldefinition aus der Datenmodellablage in die PostGIS-Datenbank geladen und als neues Datenschema konfiguriert. Das Programm setzt die Modellelemente wie folgt um:

- Klassen und Strukturen: Pro Objekt eine Tabelle, Einführung Tabellenidentifikator `t_id` als Primärschlüssel plus Sequenzattribut `t_seq` für die Reihenfolge der Strukturelemente.

```
CREATE TABLE …_b (
  t_id integer PRIMARY KEY,
  t_ili_tid uuid DEFAULT uuid_generate_v4(),
  b numeric(10,3) NOT NULL,
  zu_a integer,
  CONSTRAINT …_zu_a_fkey FOREIGN KEY (zu_a)
   REFERENCES …_a_erweitert (t_id) MATCH SIMPLE
   ON UPDATE NO ACTION ON DELETE NO ACTION
   DEFERRABLE INITIALLY DEFERRED
);
```

```
CREATE TABLE ….struct (
  t_id integer PRIMARY KEY,
  t_seq integer NOT NULL,
  s character varying(100) NOT NULL,
  …_B_s integer,
  CONSTRAINT …_b_s_fkey FOREIGN KEY (b_s)
   REFERENCES …_b (t_id) MATCH SIMPLE
   ON UPDATE NO ACTION ON DELETE NO ACTION
   DEFERRABLE INITIALLY DEFERRED
);
```

- Vererbung: Hier kommt die SubClass-Strategie zur Anwendung: die Eigenschaften der abstrakten Mutter-Klasse `a` werden in die erweiterte Klasse `a_erweitert` geschrieben.

```
CREATE TABLE …_a_erweitert (
  t_id integer PRIMARY KEY,
  t_ili_tid uuid DEFAULT uuid_generate_v4(),
  t character varying(255),
  a character varying(24) NOT NULL
);
```

  Falls die NewClass-Strategie zur Anwendung kommt (vgl. Dokumentation ili2pg), wird jede Klasse in eine Tabelle abgebildet, was die Aufteilung der Objektinstanzen auf mehrere Tabellen zur Folge hat. Die Oberklasse erhält jeweils ein Attribut `t_type`, das den Klassennamen der Unterklasse enthält.

- Assoziationen und Referenzattribute: Automatische Erzeugung der oben beschriebenen Konsistenzbedingungen für Fremdschlüssel über die Rolle `zu_a` der Assoziation.

```
CREATE TABLE …_a_erweitert (
  t_id integer PRIMARY KEY,
  t_ili_tid uuid DEFAULT uuid_generate_v4(),
  t character varying(255),
  a character varying(24) NOT NULL
);
```

```
CREATE TABLE …_b (
  t_id integer PRIMARY KEY,
  t_ili_tid uuid DEFAULT uuid_generate_v4(),
  b numeric(10,3) NOT NULL,
  zu_a integer,
  CONSTRAINT …_zu_a_fkey FOREIGN KEY (zu_a)
   REFERENCES …_a_erweitert (t_id) MATCH SIMPLE
   ON UPDATE NO ACTION ON DELETE NO ACTION
   DEFERRABLE INITIALLY DEFERRED
);
```

- Strukturattribute: Erzeugung eines Fremdschlüssels in der Struktur-Tabelle. In der Tabelle `b` wird das Strukturattribut `s` nicht aufgeführt. Durch die Referenzierung eines konkreten Klassenobjekts aus der Struktur heraus wird die Zuordnung realisiert.

```
CREATE TABLE …_b (
  t_id integer PRIMARY KEY,
  t_ili_tid uuid DEFAULT uuid_generate_v4(),
  b numeric(10,3) NOT NULL,
  zu_a integer,
  CONSTRAINT …_zu_a_fkey FOREIGN KEY (zu_a)
   REFERENCES …_a_erweitert (t_id) MATCH SIMPLE
   ON UPDATE NO ACTION ON DELETE NO ACTION
   DEFERRABLE INITIALLY DEFERRED
);
```

```
CREATE TABLE ….struct (
  t_id integer PRIMARY KEY,
  t_seq integer NOT NULL,
  s character varying(100) NOT NULL,
  …_B_s integer,
  CONSTRAINT …_b_s_fkey FOREIGN KEY (b_s)
   REFERENCES …_b (t_id) MATCH SIMPLE
   ON UPDATE NO ACTION ON DELETE NO ACTION
   DEFERRABLE INITIALLY DEFERRED
);
```

- Aufzählungen: Hier werden Aufzähltypen so umgesetzt, dass eine zusätzliche Tabelle mit den Aufzählungselementen erzeugt wird. Sie erhält den Namen des Aufzählattributs. Dies erleichtert die Datenbewirtschaftung, weil alle zulässigen Elemente in der Datenbank vor-

handen sind. In der Tabelle der Klasse wird ein normales Textattribut erzeugt, welches die Werte der Aufzählung aufnimmt.

```
CREATE TABLE …_a_erweitert (
  t_id integer PRIMARY KEY,
  t_ili_tid uuid DEFAULT uuid_generate_v4(),
  t character varying(255),
  a character varying(24) NOT NULL
);
```

```
CREATE TABLE …_type (
  itfcode integer PRIMARY KEY,
  ilicode character varying(1024) NOT NULL,
  seq integer,
  dispname character varying(250) NOT NULL,
);
```

Aufzähltyp-Tabellen werden wahlweise zusätzlich als Text (`ilicode`) importiert:

itfcode [PK] integer	ilicode character varying(1024)	seq integer	dispname character varying(250)
0	ta		ta
1	tb		tb
2	tc		tc

- Objektidentifikatoren (OID): Wenn im Modell eine OID-Deklaration vorhanden ist, soll diese Eigenheit auch in der Datenbank konfiguriert werden. Hier soll gelten, dass alle Objekte einen OID vom Typ UUID erhalten. Solche OID werden durch ein zusätzliches Attribut realisiert. Strukturen und Aufzählungstabellen erhalten keine OID, da sie keine selbstständigen Objekte sind.

```
CREATE TABLE …_a_erweitert (
  t_id integer PRIMARY KEY,
  t_ili_tid uuid DEFAULT uuid_generate_v4(),
  t character varying(255),
  a character varying(24) NOT NULL
);
```

- Zusätzlich werden einige Metadatentabellen angelegt, die ili2pg vor allem für den Datenexport benötigt. Diese Tabellen müssen nicht aktiv bewirtschaftet werden.

Schritt 2 – Produktivdaten in die Modellstruktur umbauen

Beim Datenumbau werden die Objektinstanzen aus der Produktivdatenbank in die Modellstruktur überführt. Wenn die Produktion auf die Modellstruktur umgestellt wird, muss dieser Schritt nur einmal ausgeführt werden. Falls weiterhin in der gegebenen Produktivdatenbank gearbeitet wird, so ist der Datenumbau bei Bedarf, regelmässig oder sogar automatisiert durchzuführen. Der Datenumbau findet innerhalb der PostGIS-Datenbank statt.

Im kantonalen Raumdatenpool wird der Datenumbau im Rahmen des ETL-Prozesses mittels SQL-Skripten in der Datenbank abgewickelt.

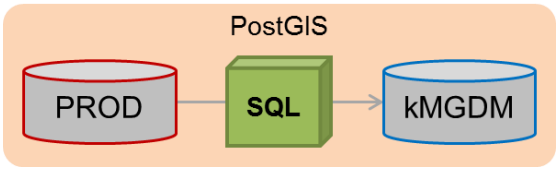

Abb. 7: Datenumbau mit SQL

Schritt 3 – Transferdaten exportieren

Die modellkonformen PostGIS-Daten werden mit ili2pg als INTERLIS 2-XML-Transferformat exportiert. Die Modellelemente werden gemäss INTERLIS 2.3 Referenzhandbuch, Kapitel 3, kodiert. Alternativ können Transferdaten als INTERLIS 2-GML-Transferformat gemäss eCH-0118 exportiert werden.

Schritt 4 – Transferdaten prüfen und nutzen

Um die Qualität der Exportdaten sicherzustellen, wird der exportierte Datensatz im *Checker für INTERLIS 2* gegenüber dem Datenmodell geprüft. Bei dieser Prüfung wird nicht nur die Syntax überprüft, sondern auch der Inhalt: kommen ungültige Aufzählungswerte vor? Stimmt die Geometrie? Sind alle Beziehungsreferenzen korrekt? Nach erfolgreicher Prüfung können die Daten weitergegeben und genutzt werden.

Anwendungsbeispiele

Gebäudedatenmodell «Buildings_V1»

Ein fiktives Modellbeispiel für eine *Gebäudebewirtschaftung* wird umgesetzt, mit Daten versehen und aus der Datenbank exportiert. Dieses Modellbeispiel wird im Rahmen der ili2pg-Workshops verwendet und wurde leicht modifiziert.

Im Modell werden Wohn- und Verwaltungsgebäude bewirtschaftet. Gebäude haben die allgemeine Eigenschaften Geschosszahl (*Storeys*), Dachform (*Roof*), Adresse und Geometrie. Wohngebäude haben zusätzlich die Anzahl Wohnungen. Die Adressen werden strukturiert und die Verwaltungsgebäude haben eine Zugehörigkeit zu einer Verwaltungseinheit. Die Gebäudegeometrie ist – mehr Demonstrationszwecken geschuldet denn praktischer Plausibilität – als Multifläche definiert.

Schritt 0 – UML-Datenmodell (Modell Buildings_V1, Topic Buildings):

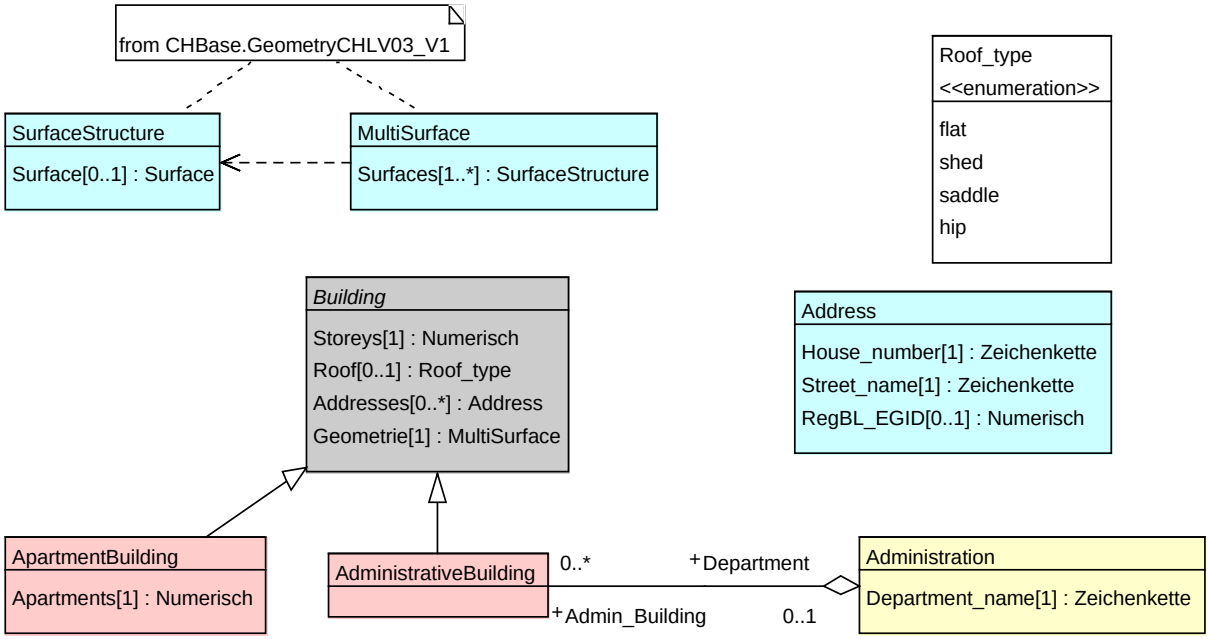

Abb. 8: Datenmodell Buildings_V1

Das Modell vereinigt die meisten typischen Modellkonstrukte: Aufzählung, Strukturattribut, Vererbung, Beziehung, Import von CHBase-Modulen.

Zusätzlich gilt die Konvention, dass alle Objekte Identifikatoren vom Typ UUID erhalten. Dies ist im Modell deklariert und am besten in der `*.ili`-Datei ersichtlich:

```
…
TOPIC Buildings =
  OID AS INTERLIS.UUIDOID;
  …
```

Schritt 1 – ili2pg-Befehl zur PostGIS-Konfiguration mit dem Datenmodell:

`java -jar ili2pg.jar`	Programmaufruf
`--schemaimport`	Ein neues Schema konfigurieren
`--dbhost MY_HOST --dbusr MY_USR --dbpwd MY_PWD --dbdatabase MY_DB`	
`--dbschema _buildings`	Name des neuen Datenbankschemas
`--importTid`	Abbildung der OID als UUID
`--nameByTopic`	Qualifizierung der Tabellennamen
`--createEnumTabs`	Eigene Tabellen für die Aufzählltypen
`--smartInheritance`	«Smarte» Abbildung der Vererbung
`--coalesceMultiSurface`	MultiSurface direkt als Geometrietyp[1]
`--createGeomIdx`	Räumlichen Index erzeugen
`--createFk`	Fremdschlüssel-Konsistenzbedingungen
`--createscript Buildings_V1_create.sql`	SQL-Create-Skript speichern
`./Buildings_V1.ili`	Modelldatei

Schritt 2 – Datenumbau mit SQL: nicht anwendbar, da neue Daten originär in der Modellstruktur erfasst werden. Ein SQL-Skript zur Erzeugung der nachfolgend dargestellten Testdaten kann von der Webseite des kantonalen Geoportals heruntergeladen werden.

Die Attributbildung wird wie folgt erledigt:

buildings_roof_type (Aufzählltyp)	
`itfcode`	Aufzählltyp gemäss INTERLIS 1-Codierung: der erste Eintrag hat die Nummer 0, der zweite Eintrag die Nummer 1 etc.; Primärschlüssel
`ilicode`	Aufzählltyp gemäss INTERLIS 2-Codierung: Einträge gemäss Modelldefinition als Text
`seq`	Reihenfolge der Aufzählelemente – hier ohne Belang.
`dispname`	Vgl. `ilicode`

buildings_address (Struktur)	
`t_id`	Laufender Tabellenidentifikator als Primärschlüssel
`t_seq`	Reihenfolge der Strukturelemente
(spezifische Attribute)	–
`buildngs_dmstrtvblding_addresses`	Erzeugter Fremdschlüssel → Verwaltungsgebäude, zu dem die entsprechende konkrete Adresse gehört
`buildngs_prtmntblding_addresses`	Erzeugter Fremdschlüssel → Wohngebäude, zu dem die entsprechende konkrete Adresse gehört

[1] MultiSurface aus CHBase beinhaltet Kreisbogen. Wenn nur Geraden zulässig sein sollen, kann die Option `--strokeArcs` verwendet werden. Dann wird MultiPolygon als Geometrietyp angelegt.

buildings_administration – Tabelle der Verwaltungseinheiten	
t_id	Laufender Tabellenidentifikator als Primärschlüssel
t_ili_tid	Objektidentifikatoren (OID) vom Typ UUID inklusive Funktion zum automatischen Generieren der OID
(spezifische Attribute)	–

buildings_administrativebuilding – Tabelle der Verwaltungsgebäude	
t_id	Laufender Tabellenidentifikator als Primärschlüssel
t_ili_tid	Objektidentifikatoren (OID) vom Typ UUID inklusive Funktion zum automatischen Generieren der OID
department	Fremdschlüsselattribut zur Realisierung der Beziehung auf die Klasse/Tabelle buildings_administration. Das Attribut ist vom Typ integer und bildet zusammen mit der Fremdschlüssel-Konsistenzbedingung die Beziehung ab.
geometrie	Geometrie vom Typ MultiSurface bzw. MultiPolygon
(spezifische Attribute)	–

buildings_apartmentbuilding – Tabelle der Wohngebäude	
t_id	Laufender Tabellenidentifikator als Primärschlüssel
t_ili_tid	Objektidentifikatoren (OID) vom Typ UUID inklusive Funktion zum automatischen Generieren der OID
geometrie	Geometrie vom Typ MultiSurface bzw. MultiPolygon
(spezifische Attribute)	–

Hinweis zur Geometrie: Die Tabellen multisurface und surfacestructure aus der Strukturdefinition von CHBase werden gemäss Modell in jedem Fall angelegt. Normalerweise müssen diese Tabellen nicht bewirtschaftet werden; es kann jedoch spezielle Fälle geben, bei denen nicht direkt eine MultiSurface- bzw. eine MultiPolygon-Geometrie angelegt werden kann. Dann müssen die Geometrien über diese beiden Tabellen gebildet werden.

Schritt 3 – ili2pg-Befehl für den Datenexport im INTERLIS 2-XML-Transferformat:

```
java -jar ili2pg.jar                                  Programmaufruf
 --export                                             Transferdaten exportieren
 --dbhost MY_HOST --dbusr MY_USR --dbpwd MY_PWD --dbdatabase MY_DB
 --dbschema _buildings                                Name des neuen Datenbankschemas
 --models Buildings_V1                                Modellname
 ./buildings.xtf                                      Transferdatei im INTERLIS 2-XML-Format
```

Ganz leicht können anstelle von INTERLIS 2-XML-Transferdaten auch INTERLIS-GML-Transferdaten gemäss eCH-0118 exportiert werden:

```
…
./buildings.gml                                       Transferdatei im INTERLIS GML-Format
```

In der exportierten Transferdatei werden die UUID-OID als TID gespeichert! Auf diese Weise kann die Objektintegrität auch über Datenexport und Datenimport gewährleistet werden und beispielsweise in eine Historisierung überführt werden.

Struktur der Exportdatei im Format INTERLIS 2-XML (XTF):

```
<?xml version="1.0" encoding="UTF-8"?><TRANSFER xmlns="http://www.interlis.ch/INTERLIS2.3">
<HEADERSECTION SENDER="ili2pg-3.0.1-20160126" VERSION="2.3">
…
</HEADERSECTION>
<DATASECTION>
  <Buildings_V1.Buildings BID="Buildings_V1.Buildings">
    <Buildings_V1.Buildings.Administration TID="8a1e4ca2-4991-48e7-9c25-a0b2defd4618">
      <Department_name>Departement Bau und Umwelt</Department_name>
    </Buildings_V1.Buildings.Administration>
    …
    <Buildings_V1.Buildings.AdministrativeBuilding TID="b47a8ac3-2e09-4729-9d75-5d1a21d5fa51">
      <Storeys>4</Storeys>
      <Roof>hip</Roof>
      <Addresses>
        <Buildings_V1.Buildings.Address>
          <House_number>2</House_number>
          <Street_name>Kirchstrasse</Street_name>
          <RegBL_EGID>2389084</RegBL_EGID>
        </Buildings_V1.Buildings.Address>
      </Addresses>
      <Geometrie>
        <GeometryCHLV03_V1.MultiSurface>
          <Surfaces>
            <GeometryCHLV03_V1.SurfaceStructure>
              <Surface>
                <SURFACE>
                  <BOUNDARY>
                    <POLYLINE>
                      <COORD><C1>723819.197</C1><C2>211313.882</C2></COORD> …
                    </POLYLINE>
                  </BOUNDARY>
                </SURFACE>
              </Surface>
            </GeometryCHLV03_V1.SurfaceStructure>
          </Surfaces>
        </GeometryCHLV03_V1.MultiSurface>
      </Geometrie>
      <Department REF="8a1e4ca2-4991-48e7-9c25-a0b2defd4618"></Department>
    </Buildings_V1.Buildings.AdministrativeBuilding>
    …
    <Buildings_V1.Buildings.ApartmentBuilding TID="091faa34-119b-4329-bd6d-bc86d827af54">
      <Storeys>4</Storeys>
      <Roof>hip</Roof>
      <Addresses>
        <Buildings_V1.Buildings.Address>
          <House_number>36</House_number>
          <Street_name>Bankstrasse</Street_name>
          <RegBL_EGID>293130</RegBL_EGID>
        </Buildings_V1.Buildings.Address>
      </Addresses>
      <Addresses>
        <Buildings_V1.Buildings.Address>
          <House_number>6</House_number>
          <Street_name>Rathausgasse</Street_name>
          <RegBL_EGID>293130</RegBL_EGID>
        </Buildings_V1.Buildings.Address>
      </Addresses>
      <Geometrie> … </Geometrie>
      <Apartments>8</Apartments>
    </Buildings_V1.Buildings.ApartmentBuilding>
    …
  </Buildings_V1.Buildings>
</DATASECTION>
</TRANSFER>
```

Andere Transferdatenformate

Für den in der Einleitung erwähnten Datenbezug dürften INTERLIS 2-XML-Daten nur in zweiter Linie beziehungsweise vor allem von «offiziellen Stellen» nachgefragt werden. In Sinne einer nutzerorientierten Datenbereitstellung müssen weitere Abgabeformate aus den modellkonformen Daten abgeleitet werden. Dazu können Formate wie GeoPackage (GPKG) GML oder auch CSV gehören. Für GPKG steht ebenfalls ein generisches Schnittstellenwerkzeug zur Verfügung, *ili2gpkg*. Damit können INTERLIS-Transferdaten mit zugehörigem Datenmodell einfach in GPKG umgewandelt werden. Der oben beschriebene Schritt 1 kann direkt mit dem Import der INTERLIS 2-XML-Transferdaten kombiniert werden:

`java -jar ili2gpkg.jar`	Programmaufruf
`--import`	Ein neues GPKG konfigurieren und vorhandene Transferdaten importieren
`--dbfile ./buildings.gpkg`	Neues GPKG anlegen
`--importTid`	Abbildung der OID als UUID
`--nameByTopic`	Qualifizierung der Tabellennamen
`--coalesceMultiSurface`	MultiSurface als Geometrietyp
`--strokeArcs`	GPKG kann nicht mit Kreisbogen umgehen
`--createGeomIdx`	Räumlichen Index erzeugen
`./buildings.xtf`	Transferdaten (Input) als INTERLIS 2-XML

Datenmodell mit externem Katalog «Energiefoerderung_V1»

Das kantonale Datenmodell der Energieförderung besitzt zwei Kataloge, für die *Gebäudekategorie* und für die (Energieförderungs-)*Massnahmen*. Das UML-Klassendiagramm ist nachfolgend dargestellt (Schritt 0):

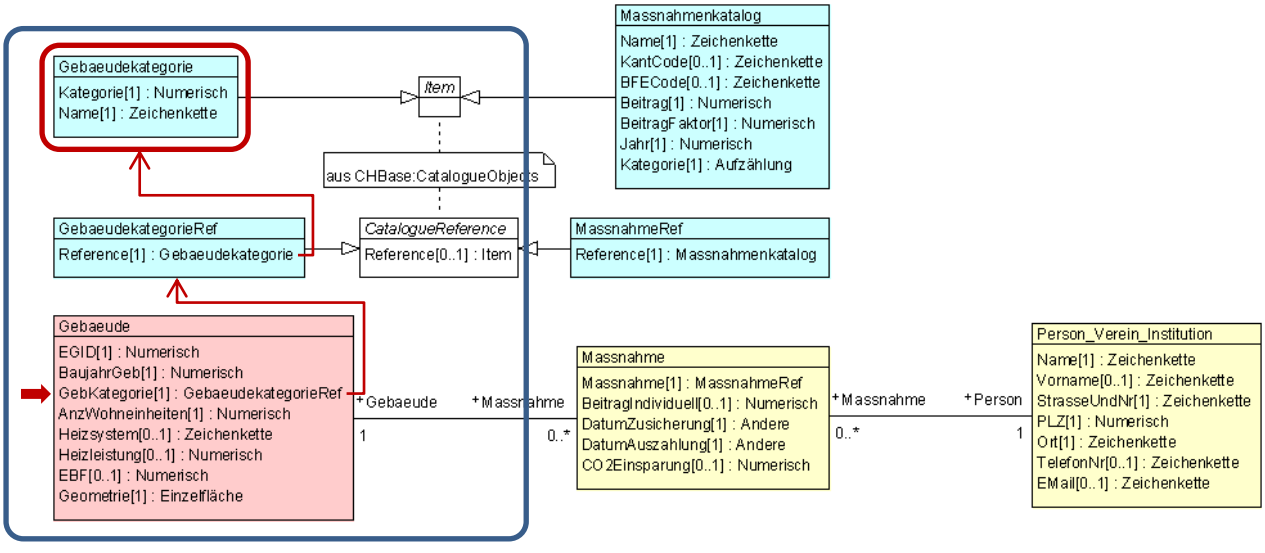

Abb. 9: Datenmodell Energiefoerderung_V1

Die Kataloge werden im Datenmodell über Referenzierungs-Strukturen eingebunden. Der wesentliche Vorteil von Katalogen gegenüber gewöhnlichen Aufzählungen ist, dass die zulässigen Werte nicht im Modell hart codiert sind, sondern als Daten vorliegen, welche jederzeit bearbeitet werden können[2]. Die Umsetzung von Katalogen wird am Beispiel der Gebäudekategorie erläutert (in Abb. 9 blau umrandet). Zum besseren Verständnis wird nachfolgend der entsprechende Ausschnitt aus der INTERLIS 2.3-Modelldefinition abgebildet.

```
…
TOPIC Energiefoerderung_Kataloge =

  CLASS Gebaeudekategorie
  EXTENDS CatalogueObjects_V1.Catalogues.Item =
    Kategorie : MANDATORY 0 .. 99;
    Name : MANDATORY TEXT*255;
  END Gebaeudekategorie;

  STRUCTURE GebaeudekategorieRef
  EXTENDS CatalogueObjects_V1.Catalogues.CatalogueReference =
    Reference (EXTENDED) :
      MANDATORY REFERENCE TO (EXTERNAL) ….Gebaeudekategorie;
  END GebaeudekategorieRef;
…
END Energiefoerderung_Kataloge;

TOPIC Energiefoerderung =
  DEPENDS ON Energiefoerderung_V1.Energiefoerderung_Kataloge;

  CLASS Gebaeude =
    EGID : MANDATORY 100000 .. 999999999;
    GebKategorie : MANDATORY ….GebaeudekategorieRef;
    … (weitere Klassenattribute)
    Geometrie : MANDATORY SURFACE …;
  END Gebaeude;
…
END Energiefoerderung;
…
```

Die Klasse `Gebaeudekategorie` enthält als Daten die Katalogeinträge. Das Referenzattribut `Reference` verweist auf einen solchen Katalogeintrag. Gemäss INTERLIS 2.3-Sprachregeln können Referenzattribute nur in Strukturen definiert werden, weshalb die Struktur `GebaeudekategorieRef` eingeführt werden muss. In der Klasse `Gebaeude` wird nun das Attribut `GebKategorie` vom Typ `GebaeudekategorieRef` definiert, so dass schliesslich einem konkreten `Gebaeude`-Objekt die zutreffende Kategorie aus dem Katalog zugeordnet werden kann.

[2] Eine ausführliche Dokumentation ist unter http://www.geo.admin.ch → Geodaten → Geobasisdaten → Geodatenmodelle → Basismodule des Bundes… als PDF aufgeschaltet.

Schritt 1 – ili2pg-Befehl zur PostGIS-Konfiguration mit dem Datenmodell:

```
java -jar ili2pg.jar                                    Programmaufruf
 --schemaimport                                         Ein neues Schema konfigurieren
 --dbhost MY_HOST --dbusr MY_USR --dbpwd MY_PWD --dbdatabase MY_DB
 --dbschema _energie                                    Name des neuen Datenbankschemas
 --nameByTopic                                          Qualifizierung der Tabellennamen
 --createEnumTabs                                       Eigene Tabellen für die Aufzählltypen
 --smartInheritance                                     «Smarte» Abbildung der Vererbung
 --coalesceCatalogueRef                                 direkte Abbildung der Katalogstruktur
 --coalesceMultiSurface                                 MultiSurface direkt als Geometrietyp
 --createGeomIdx                                        Räumlichen Index erzeugen
 --createFk                                             Fremdschlüssel-Konsistenzbedingungen
 --models Energiefoerderung_V1                          Modell (wird aus Repository geladen)
```

Dank der Programmoption `--coalesceCatalogueRef` werden die Kataloge «smart» in der Datenbank implementiert. Das bedeutet, dass die Referenz auf den Katalog (Tabelle `Gebaeudekategorie`) direkt in der Tabelle `Gebaeude` als Fremdschlüsselattribut mit einer Fremdschlüssel-Konsistenzbedingung eingeführt wird:

```
CREATE TABLE ….energfrdng_ktloge_gebaeudekategorie
(
  t_id integer PRIMARY KEY,
  kategorie integer NOT NULL,
  aname character varying(255) NOT NULL,    Beachte: der Attributname «name» wird
                                            automatisch in «aname» umbenannt, um
                                            Konflikte mit dem in PostgreSQL/PostGIS
                                            reservierten Wort «name» zu vermeiden.
)…;

CREATE TABLE ….energiefoerderung_gebaeude
(
  t_id integer PRIMARY KEY,
  egid integer NOT NULL,
  gebkategorie integer NOT NULL,            Fremdschlüssel auf den Katalog
  …
  geometrie geometry(CurvePolygon,21781),
  CONSTRAINT …_enrgfrdng_ktloge_gebaeudekategorie_fkey
    FOREIGN KEY (gebkategorie)
    REFERENCES ….energfrdng_ktloge_gebaeudekategorie
    MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION
    DEFERRABLE INITIALLY DEFERRED
)…;
```

Bei der Datenbewirtschaftung im Zusammenhang mit Katalogen ist zu beachten, dass externe Datensätze zu Katalogen zuerst in die Datenbank importiert werden müssen und danach die konkreten Datenobjekte (`Gebaeude`) erzeugt werden. Damit kann bei der Erzeugung der Datenobjekte direkt die zutreffende Katalogreferenz eingefügt werden.

Weiterführende Aspekte

Das Projekt «Modellkonformer Austausch von Geodaten» (MDX) in Zusammenarbeit von IKGEO und GKG/KOGIS erörtert die Bedeutung des Begriffs «Download-Dienst» im Sinne der Geoinformationsverordnung (GeoIV) sowie die technischen Voraussetzungen und Möglichkeiten, solche Download-Dienste modellkonform zu realisieren.

In diesem Zusammenhang steht neben dem INTERLIS 2-XML-Transferformat auch GML gemäss *eCH-0118 GML-Kodierungsregeln für INTERLIS* im Vordergrund, insbesondere bei der Umsetzung der Download-Dienste als OGC WFS.

ANHANG – Nutzung des Gebäudedatenmodells in QGIS

Hier wird die Nutzung der Daten des Gebäudedatenmodells aus PostGIS in QGIS erläutert. Ein SQL-Skript zur Erzeugung der nachfolgend dargestellten Testdaten kann von der Webseite des kantonalen Geoportals heruntergeladen werden.

Erster Schritt: Layer in QGIS laden

Abb. 10: Layer des Gebäudedatenmodells in QGIS laden

Zweiter Schritt: Zuordnung [1–m] der Adress-Struktur zu den Gebäuden realisieren

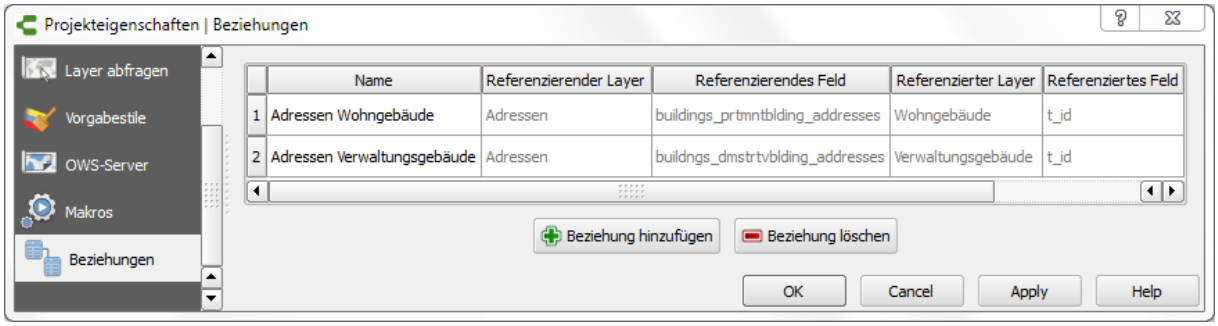

Abb. 11: Zuordnung der Adress-Struktur via Projekteigenschaften → Beziehungen

Dritter Schritt: Beziehung Verwaltungsgebäude – Verwaltungsbehörde [1–1] realisieren

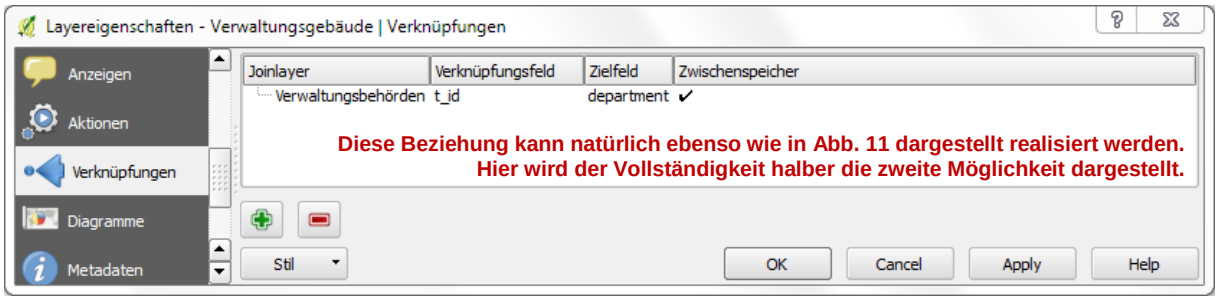

Abb. 12: Realisierung der Modellbeziehung via Layereigenschaften → Verknüpfungen

Vierter Schritt: Informationen zu Wohngebäude und zu Verwaltungsgebäude abfragen

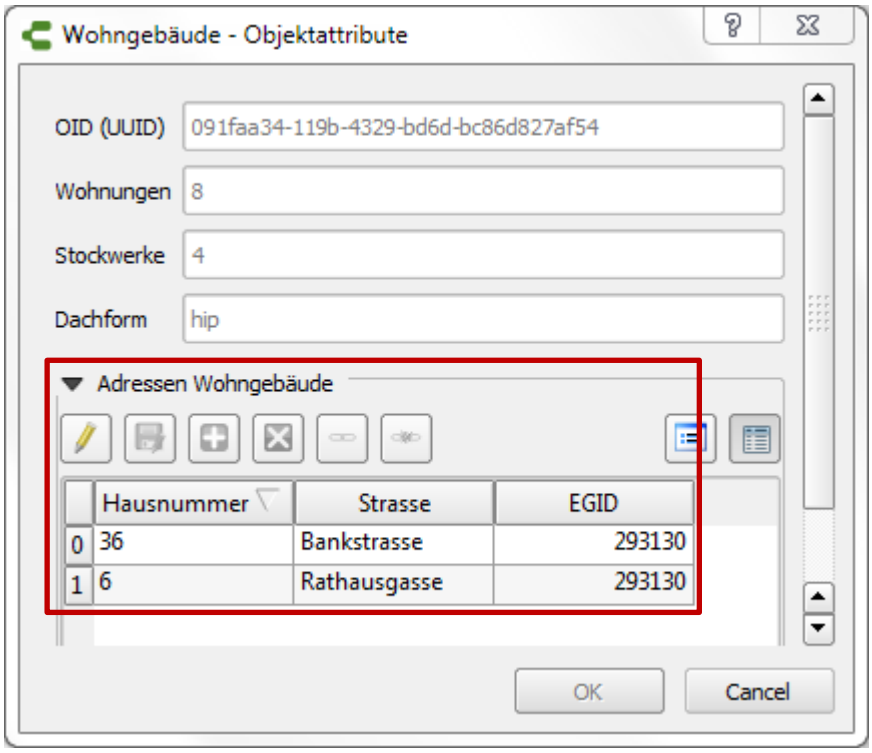

Abb. 13: Informationen zu einem Wohngebäude mit zwei Adressen

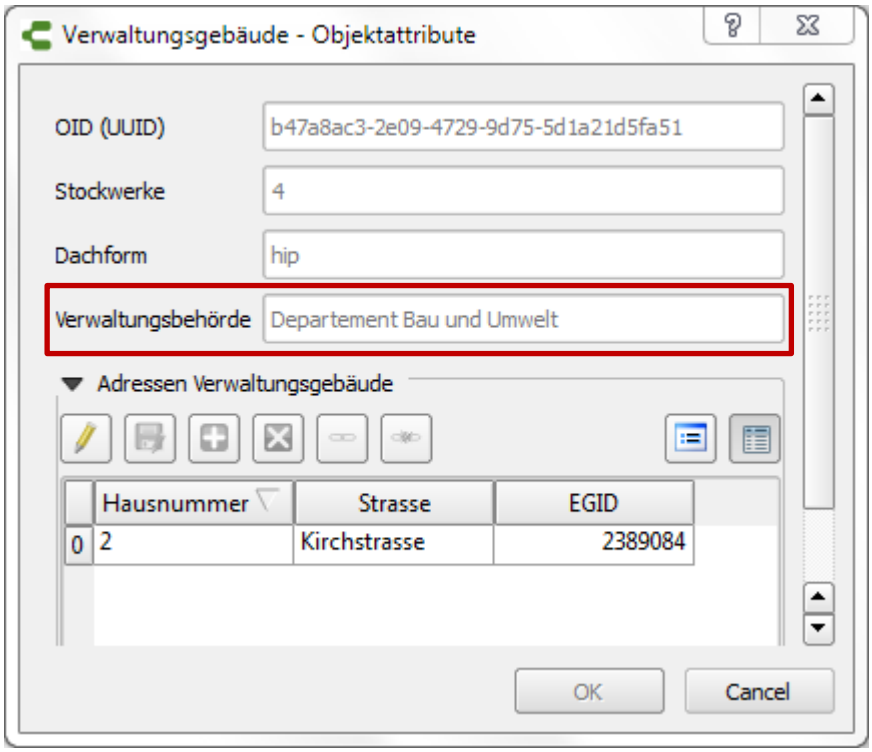

Abb. 14: Informationen zu einem Verwaltungsgebäude inklusive Zuordnung der Verwaltungsbehörde