Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Eidgenössisches Departement für Verteidigung, Bevölkerungsschutz und Sport VBS

Bundesamt für Landestopografie swisstopo

Umsetzung von konzeptionellen Geo-datenmodellen

Am Beispiel von INTERLIS 2 Modellen und SQL

Best-Practice Beispiele

Erstellt: Mai 2014

Auftraggeber:
KOGIS, Bundesamt für Landestopografie swisstopo

Autoren:
Inser SA & Eisenhut Informatik AG

Inhaltsverzeichnis

1 Einleitung

Diese Sammlung an Umsetzungsbeispielen ist als Best-Practice zu verstehen und soll eine Hilfestellung sein, wenn man vom konzeptionellen, objektorientierten Modell in INTERLIS 2 zu einem vereinfachten, relationalen Modell in SQL kommen möchte. Die Motivation dazu ergab sich einerseits aus den Erfahrungen diverser Umsetzungsprojekte bei KOGIS und dem BAFU. Andererseits nimmt swisstopo gemäss GeoIG/GeoIV die Aufgabe wahr, technische und qualitative Anforderungen an die Geobasisdaten des Bundesrechts zu stellen, insbesondere an die Modellierung.
Die Liste mit typischen Konstrukten ist nicht abschliessend und kann beliebig erweitert werden. Ebenso kann man sich für jedes Konstrukt mehrere Umsetzungsmöglichkeiten vorstellen. Hier war der Anspruch, dass einige gängige Möglichkeiten aufgezeigt werden. In diesem Sinn soll dies ein „lebendes" Dokument sein.

2 Generelle Hinweise

Ein konzeptionelles Modell dient der Dokumentation und der Harmonisierung von Geodaten. Es dokumentiert die Struktur sowie die Semantik eines Geodatensatzes systemunabhängig. Möchte man ein konzeptionelles Modell in logische Datenbankensysteme umsetzen, muss man sich vorab einige allgemeine Gedanken machen. Es ist beispielsweise abzuklären was der Zweck der logischen Anwendung ist und welche fachlichen Anforderungen zu erfüllen sind. Deshalb muss man sich generell von der Vorstellung lösen, dass ein konzeptionelles, objektorientiertes Modell 1:1 in ein logisches Modell umwandelbar ist.
Das vorliegende Dokument soll kritische Punkte der Umsetzung anhand von Beispielen erläutern. Es soll dabei nicht ausschliesslich auf nur eine Datenbanktechnologie Rücksicht genommen werden; vielmehr werden Beispiele und Ratschläge gegeben, welche Punkte beachtet werden sollen.
Es gibt meist mehrere Möglichkeiten ein konzeptionelles Modell in ein konkretes Datenbankmodell umzuwandeln. Eine wichtige Frage ist daher, wie ein implementiertes Modell gebraucht werden wird. Macht es z.B. Sinn eine flache Struktur zu verwenden, oder eine höher modellierte Datenbank mit vielen Tabellen und Bedingungen zu erstellen? Welche Personen oder Prozesse (Nutzergruppe) und mit welcher Software werden die Datenbank benutzen? Es empfiehlt sich auch einen Fachspezialisten aus der Nutzergruppe beizuziehen, der überprüft, ob eine erstellte Struktur auch aus vorgesehener Nutzersicht benutzbar ist.
Die Technologien, welche für die Implementierung der Datenbank verwendet werden, spielen ebenfalls eine grosse Rolle bei der Implementierung. Einige Datenbanksysteme ermöglichen es harte Bedingungen zu implementieren; d.h. dass es unmöglich ist, nicht konforme Daten in der Datenbank zu speichern.
Einige generelle Hinweise zur Schemaumwandlung sind folgende Punkte:

- INTERLIS 2 ist objektorientiert. Ein INTERLIS 2 Modell muss daher entschachtelt und in ein relationales Modell umgewandelt werden; Haupt- und Fremdschlüssel müssen generiert werden, um Beziehungen abbilden zu können.

- INTERLIS 2 ermöglicht es andere Modelle oder Teile von Modellen zu importieren. Die wahre Komplexität eines INTERLIS 2 Modells kann daher grösser sein, als auf den ersten Blick ersichtlich.

- INTERLIS 2 Modelle sind in Topics und Klassen strukturiert. Viele Datenbanksysteme kennen nur Schemas und Tabellen.

- Umgekehrt unterstützt INTERLIS Multipart-Objekte nur via Strukturen, wie sie z.B. in CH-Base Datenmodellen vordefiniert sind. Gegebenenfalls müssen Datenbanken so modelliert werden, dass sie nur Singlepart-Objekte unterstützten.

- INTERLIS 2 unterstützt mehrere Geometrien pro Klasse. In den meisten GIS-Datenbanksystemen existieren diese Konstrukte nicht und eine Klasse mit zwei Geometrien muss daher meistens in zwei oder drei Tabellen aufgeteilt werden.

- INTERLIS 2 unterstützt Strukturattribute. In den meisten handelsüblichen Datenbanksystemen und GIS existieren diese Konstrukte nicht, und eine Klasse mit Strukturattributen muss daher meistens in zwei oder mehr Tabellen aufgeteilt werden.

- INTERLIS 2 unterstützt Bedingungen innerhalb einer Klasse; z.B. dass ein Attribut grösser oder kleiner als ein anderes Attribut derselben Klasse sein muss. In vielen Datenbanksystemen sind solche Bedingungen nur mit Schwierigkeit modellierbar.

- INTERLIS 2 unterstützt Aufzählungen/kodierte Domains; nicht alle Datenbanksysteme unterstützten diese Konstrukte. Eine Möglichkeit Aufzählungen umzusetzen sind z.B. Referenztabellen.

Des Weiteren soll auch erwähnt werden, dass eine Datenbank, welche modellkonforme Daten aufnimmt nicht unbedingt selber modellkonform aufgebaut sein muss. Eine flache Struktur kann z.B. legitim sein. Es soll deshalb auch darauf hingewiesen werden, dass es möglich ist viele Bedingungen über Import- und Exportprozesse abzufangen.

3 Wichtige Punkte der Schemaumwandlung

3.1 Umwandlung von objektorientierten Strukturen

Hinweis: die Nummerierung erlaubt die Zuordnung eines Use Cases (gleiches Beispiel, Spalten 2 und 4) zu mehreren Vereinfachungsmöglichkeiten (Spalte 5).
PK = primary key
FK = foreign key

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
1.a	Vererbung	1:1	CLASS A (ABSTRACT) = Attribut_1; END A; CLASS B EXTENDS A = Attribut_2; END B;	CREATE TABLE A (ID PK, Attribut_1) CREATE TABLE B (ID PK, Attribut_2)	Für jede Klasse wird eine Tabelle erstellt.
1.b	Vererbung	Superclass	CLASS A (ABSTRACT) = Attribut_1; END A; CLASS B EXTENDS A = Attribut_2; END B;	CREATE TABLE A (ID PK, Attribut_1, Attribut_2)	Gleiches Beispiel wie Nummer 1.a. Dieses Mal wird eine Tabelle für jede Wurzelklasse erstellt. Beschränkung: wenn viele Tabellen auf die Wurzelklasse referenziert werden.
1.c	Vererbung	Subclass	CLASS A (ABSTRACT) = Attribut_1; END A; CLASS B EXTENDS A = Attribut_2; END B;	CREATE TABLE B (ID PK, Attribut_1, Attribut_2)	Gleiches Beispiel wie Nummer 1.a. Eine Tabelle wird für jede konkrete Klasse erstellt.
2.a	Strukturen	Fremdschlüssel bei Struktur	STRUCTURE A =	CREATE TABLE A (	Die Struktur ist als eine Tabelle modelliert,

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
			Attribut_1 ; Attribut_2 ; END A; CLASS B = Attribut_3: A; END B;	ID PK, B_Attribut_3, Attribut_1, Attribut_2, FOREIGN KEY B_Attribut_3_FK (B_Attribut_3) REFERENCES B (ID)) CREATE TABLE B (ID PK)	die einen Fremdschlüssel enthält, um die Verbindung mit der Haupttabelle B zu machen. Wenn das Strukturattribut ein LIST/BAG OF ist, braucht es noch ein Index-Feld in der Tabelle der Struktur.
2.b	Strukturen	Fremdschlüssel bei Strukturelement in Haupttabelle	STRUCTURE A = Attribut_1 ; Attribut_2 ; END A; CLASS B = Attribut_3: A; END B;	CREATE TABLE A (ID PK, Attribut_1, Attribut_2) CREATE TABLE B (ID PK, Attribut_3, FOREIGN KEY Attribut_3_FK (Attribut_3) REFERENCES A(ID))	Gleiches Beispiel wie 2.a. Der Fremdschlüssel ist nun in der Haupttabelle enthalten. Wenn ein Strukturattribut als LIST/BAG OF definiert ist, kann diese Lösung nicht verwendet werden. Es fehlt die Möglichkeit, ein Index-Feld zu definieren.
2.c	Strukturen	Strukturelemente als JSON (oder XML) in Haupttabelle	STRUCTURE A = Attribut_1 ; Attribut_2 ; END A; CLASS B = Attribut_1: A; END B;	CREATE TABLE B (ID PK, Attribut_1 CLOB)	Der Datentyp hängt von der Technologie ab, z.B. können CLOB oder TEXT verwendet werden.
2.d	Strukturen	Generischer Fremdschlüssel	STRUCTURE A = Attribut_1 ; Attribut_2 ; END A;	CREATE TABLE A (ID PK, Attribut_1, Attribut_2,	PARENTID ist der generische Fremdschlüssel auf die Tabelle der Klasse mit dem Strukturattribut. Aber da die selbe Struktur in verschiedenen Klassen für Strukturattribute

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
			CLASS B = Attribut_1: A; END B;	PARENTID, PARENT_FQNAME VARCHAR(1000)) CREATE TABLE B (ID PK)	verwendet werden kann, weiss man nicht auf welche Tabelle der Fremdschlüssel geht. Also enthält PARENT_FQNAME den qualifizierten Attributnamen des Strukturattributes ("ModelName.TopicName.ClassName.AttrName").
3	Strukturen	Strukturelemente in Haupttabelle einbetten	STRUCTURE A = Attribut_1 : (D1,D2) ; Attribut_2 : (D3,D4) ; END A CLASS B = Attribut_3 : A; Attribut_4 : A ; END B;	CREATE TABLE B (ID PK, Attribut_3_1 (D1,D2), Attribut_3_2 (D3,D4), Attribut_4_1 (D1,D2), Attribut_4_2 (D3,D4))	Damit kann man vermeiden, zu viele Tabellen zu erstellen. Beschränkung: wenn die Domains sehr gross sind. Die Aufzählungen können dann wie in den Beispielen 17 und 18 umgesetzt werden. Wenn ein Strukturattribut als LIST/BAG OF definiert ist, kann diese Lösung nicht verwendet werden.
4.a	Assoziationen	Ohne Zwischentabelle: nur bei 1:1 und 1:n, ohne DB-Constraints	CLASS A = Attribut_1; END A; CLASS B = Attribut_2; END B; ASSOCIATION C = rA -- {1..*} A; rB -- {1} B; END C;	CREATE TABLE A (ID PK, Attribut_1, rB,) CREATE TABLE B (ID PK, Attribut_2)	Die Verbindung zwischen zwei Tabellen wird mit einem Fremdschlüssel gemacht. Wird mühsam, wenn die Assoziation einer Erweiterung Attribute enthält.
4.b	Assoziationen	Ohne Zwischentabelle: nur bei 1:1 und 1:n, mit DB-Constraints	CLASS A = Attribut_1; END A; CLASS B = Attribut_2; END B; ASSOCIATION C =	CREATE TABLE A (ID PK, Attribut_1, rB, FOREIGN KEY rB_FK (rB) REFERENCES B(ID))	Die Verbindung zwischen zwei Tabellen wird mit einem Fremdschlüssel gemacht. Wird mühsam, wenn die Assoziation einer Erweiterung Attribute enthält. ESRI GDB: nicht umsetzbar.

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
			rA -- {1..*} A; rB -- {1} B; END C;	CREATE TABLE B (ID PK, Attribut_2)	
5.a	Assoziationen	Mit Zwischentabelle: bei n:m, ohne DB-Constraints	CLASS A = Attribut_1; END A; CLASS B = Attribut_2; END B; ASSOCIATION C = rA -- {1..*} A; rB -- {1..*} B; END C;	CREATE TABLE A (ID PK, Attribut_1,) CREATE TABLE B (ID PK, Attribut_2) CREATE TABLE C (rA, rB)	Die Verbindung zwischen zwei Tabellen wird mit einer Zwischentabelle gemacht.
5.b	Assoziationen	Mit Zwischentabelle: bei n:m mit DB-Constraints	CLASS A = Attribut_1; END A; CLASS B = Attribut_2; END B; ASSOCIATION C = rA -- {1..*} A; rB -- {1..*} B; END C;	CREATE TABLE A (ID PK, Attribut_1,) CREATE TABLE B (ID PK, Attribut_2) CREATE TABLE C (rA, rB, FOREIGN KEY rA_FK (rA) REFERENCES A(ID), FOREIGN KEY rB_FK (rB) REFERENCES B(ID))	Die Verbindung zwischen zwei Tabellen wird mit einer Zwischentabelle gemacht. ESRI GDB: nicht umsetzbar.

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
6	Assoziationen	Mit Löschweitergabe	CLASS A = Attribut_1; END A; CLASS B = Attribut_2; END B; ASSOCIATION C = A -- {1..*} A; B -<#> {1} B; END C;	CREATE TABLE A (ID PK, Attribut_1, Attribut_3, CONSTRAINT Attribut_3_FK FOREIGN KEY (Attribut3) REFERENCES B(ID) MATCH SIMPLE ON UPDATE NO ACTION ON DELETE CASCADE,) CREATE TABLE B (ID PK, Attribut_2)	• SQL basierte Datenbank: die nötige Einschränkungen müssen als „constraints“ definiert werden. Die Einschränkungen sind evtl. mühsam beim Importvorgang, da die Reihenfolge der Objekte/Records wichtig ist. Man kann nicht „Kinder“ importieren, wenn die „Eltern“ nicht existieren. • ESRI GDB: • ESRI GDB: Je nach Lizenz sind unterschiedliche Möglichkeiten verfügbar. Grundsätzlich können zwei RelationshipClasses erstellt werden: 1. Simple: da sie keine Einschränkungen enthalten, ist die Verbindung zur Visualisierung in ArcMap. Composite: „cascade“ Einschränkungen wenn ein Objekt gelöscht wird.
7.a	Assoziationen	Rekursives Modell in DB flachwalzen	CLASS A = Attribut_1; END A; ASSOCIATION C = rA1 -- {1..*} A; rA2 -- {1} A; END C;	CREATE TABLE A (ID PK, k0_Attribut_1, k1_ID, k1_rA2, k1_Attribut_1, k2_ID, k2_rA2, k2_Attribut_1, k3_ID, k3_rA2,	Eine rekursive Struktur wird in der Regel modelliert, um eine unendliche Objektstruktur aufnehmen zu können. Will man eine solche Struktur in eine Tabelle „flachwalzen“ muss man sich auf eine endliche Anzahl Objekte begrenzen, da eine Tabelle eine fixe Anzahl Spalten hat. Im Beispiel können maximal 5 direkte oder indirekte A Objekte mit einem Wurzel-Objekt gespeichert werden. (Entspricht Umsetzung #10 angewendet auf eine rekursive Assoziation).

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
				k3_Attribut_1, k4_ID, k4_rA2, k4_Attribut_1, k5_ID, k5_rA2, k5_Attribut_1)	Die Attributwerte sind redundant vorhanden. Der Anwender/Die Applikation muss bei der Erfassung sicherstellen, dass die mehrfach vorhandenen Daten für die identischen Objekte identische Inhalte haben.
7.b	Assoziationen	Rekursives Modell in DB als JSON (oder XML) in Haupt-tabelle flachwalzen	CLASS A = Attribut_1; END A; ASSOCIATION C = rA1 -- {1..*} A; rA2 -- {1} A; END C;	CREATE TABLE A (ID PK, k0_Attribut_1, kn_Objekte CLOB)	Statt jedes Objekt mit seinen Werten einzeln in Records zu speichern, kann das von einem Wurzelobjekt erreichbare Geflecht in einem geeigneten Format serialisiert und als ein Spaltenwert mit dem Wurzel-Objekt gespeichert werden. (Entspricht Umsetzung #2.c angewendet auf eine rekursives Ob-jektgeflecht). Abfragen mit Kriterien auf den Objekten des Geflechts sind so schwierig zu realisieren. Die Attributwerte sind redundant vorhanden. Der Anwender/Die Applikation muss bei der Erfassung sicherstellen, dass die mehrfach vorhandenen Daten für die identischen Objekte identische Inhalte haben.
8	Assoziationen	Klassen zusammen in eine Tabelle „flachwalzen“	CLASS A = Attribut_1; END A; CLASS B = Attribut_2; END B; ASSOCIATION C = rA -- {1..*} A; rB -- {1..*} B;	CREATE TABLE A (ID_CLASSA, ID_CLASSB, Attribut_1, Attribut_2)	Vorteil: Auswertungen können ohne DB-Joins gemacht werden. Nachteile: Die Attributwerte sind redundant vorhanden. Der Anwender/Die Applikation muss bei der Erfassung sicherstellen, dass die mehrfach vorhandenen Daten für die identischen Objekte identische Inhalte haben.

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
			END C;		
9	Baskets	Mehrere Baskets des selben TOPICs pro Datenbank	TOPIC A = CLASS AA = Attribut_1; END AA; END A ;	CREATE TABLE A (ID PK, Attribut_1, BasketID FOREIGN KEY BasketID _FK (BasketID) REFERENCES Basket(ID),) CREATE TABLE Basket (ID PK)	Im transferfile kann man mehrere baskets pro Topic haben. Wenn man wissen will, welche Daten zu welchem Interlis-Basket gehören, braucht man ein Extrafeld.

3.2 Klassennamen, Attributnamen

PK = primary key
FK = foreign key

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
10	Namen	Tabelle, Spalten umbenennen wegen Konflikt mit DB-Regeln oder zwingenden Vorgaben: 1. Schlüsselwörter 2. Längen 3. Klein- und Grossschreibung	1. Tabelle „Where“ 2. ESRI file GDB: max field name length: 64 characters 3. Attributename KW_Ueberschwemmung	1. Tabelle „Where_1“ 2. Falls zu lang, muss man der Name ausschneiden. 3. Attributename kw_ueberschwemmung	1. Schlüsselwörter die in Datenbanken nicht unterstützt werden (z.B. Value) 2. Längen von Attributenamen die in Datenbanken nicht unterstützt werden. 3. Grossschreibung können nicht unterstützt werden.
11	TOPICs	Mehrere TOPICs pro Modell/Datenbank	TOPIC A = CLASS AA = Attribut_1; END AA; END A ; TOPIC B = CLASS AA = Attribut_2; END AA; END B ;	CREATE TABLE A_AA (ID PK, Attribut_1) CREATE TABLE B_AA (ID PK, Attribut_2)	Weil in der DB keine TOPICs existieren, müssen für eindeutige Tabellennamen diese evtl. mit dem TOPIC-Namen ergänzt werden. Im transferfile kann man mehrere baskets haben (ein Basket pro Topic). Wenn man wissen will, welche Daten zu welchem Topic gehören, braucht man eine entsprechende Namensgebung der Tabellen.

3.3 OID, PrimaryKey

PK = primary key
FK = foreign key

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
12	OID	OID/TID als interner Schlüssel	CLASS A = Attribut_1; END A;	CREATE TABLE A (ID PK, Attribut_1)	TID ist eindeutig in einem Transferfile (.xtf), aber ist i.d.R. nicht eindeutig wenn mehrere XTF Files importiert werden. Es muss darum eine Umnummerierung stattfinden. OID ist global eindeutig.Mehrere XTF Files können ohne Umnummerierung importiert werden. Nachteil: Wenn sich das externe Nummerierungsystem ändert, müssen die internen Referenzen angepasst werden.
13	OID	OID/TID nicht als interner Schlüssel	CLASS A = Attribut_1; END A;	CREATE TABLE A (ID PK, xtf_id, Attribut_1)	Beim Import muss für alle Objekte, auch solche mit einer OID eine neue Nummer vergeben werden. Auch bei TIDs keine ID-Konflikte, da die TIDs intern nicht als PK verwendet wird. Vorteil: Wenn sich das externe Nummerierungsystem ändert, müssen die internen Referenzen nicht angepasst werden.
14.a	Verweis auf nicht bekannte Objekte (EXTERNAL)	Mit alternativem Referenzwert	CLASS A = Attribut_1; END A; CLASS B = Attribut_2; END B; ASSOCIATION C = rA -- {1..*} A; rB (EXTERNAL) -- {1} B;	CREATE TABLE A (ID PK, Attribut_1, rB, FOREIGN KEY rB_FK (rB) REFERENCES B(ID) rBext,) CREATE TABLE B (	rBext ist für die TID/OID des Objektes das unbekannt, nicht in der DB vorhanden, ist. z.B. in externen Katalogen.

			END C;	ID PK, Attribut_2)	
14.b	Verweis auf nicht bekannte Objekte (EXTERNAL)	Mit Proxy-Objekt	CLASS A = Attribut 1; END A; CLASS B = Attribut 2; END B; ASSOCIATION C = rA -- {1..*} A; rB (EXTERNAL) -- {1} B; END C;	CREATE TABLE A (ID PK, Attribut 1, rB, FOREIGN KEY rB_FK (rB) REFERENCES B(ID)) CREATE TABLE B (ID PK, IST_UNBEKANNT Attribut 2)	Die unbekannten Objekte werden in die Tabelle eingetragen, aber als unbekannt markiert (mit dem Flag IST_UNBEKANNT). Fremdschlüssel können somit wie normal auf einen Record verweisen

3.4 Grunddatentypen

PK = primary key
FK = foreign key

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
15.a	Aufzählungen	Als Name	DOMAIN D = (D1, D2, D3) CLASS A = Attribut_1: D; END A;	CREATE TABLE D (ID VARCHAR(100) PK, Description) CREATE TABLE A (ID PK, Attribut_1, CONSTRAINT Attribut_1_FK FOREIGN KEY (Attribut1) REFERENCES D(ID) MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION,)	• SQL basierte Datenbank: Tabellen mit Constraints • ESRI GDB: CodedValueDomain Name der Aufzählelemente als ID übernehmen.
15.b	Aufzählungen	Als Code	DOMAIN D = (D1, D2, D3) CLASS A = Attribut_1: D; END A;	CREATE TABLE D (ID BIGINT PK, NAME VARCHAR(100) PK, Description) CREATE TABLE A (ID PK, Attribut_1, CONSTRAINT Attribut_1_FK FOREIGN KEY (Attribut1) REFERENCES D(ID) MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION,)	• SQL basierte Datenbank: Tabellen mit Constraints • ESRI GDB: CodedValueDomain Die Codeliste kann eine Fachbedeutung enthalten. Wichtig zu beachten: wie die Integer generieren. Z. B. „1, 2, 3, 4, …“ oder „11, 12, 15, 16, …“.
16	Datentypen von Interlis die in DB	Als TEXT, CLOB oder BLOB	CLASS A = Attribut_1 : HALIGNMENT;	CREATE TABLE A (ID PK,	Datentypen können nicht direkt verwendet werden.

	nicht existieren		Attribut_2 : BLACKBOX BINARY; Attribut_3: BLACKBOX XML; END A;	Attribut_1 VARCHAR(20), Attribut_2 BLOB, Attribut_3 CLOB)	Die Hilfstabelle (Anhang 4.1) enthält die verschiedenen Attributtypen (Zahlen und Text), die in PostGreSQL und in ESRI GDB unterstützt sind.
17.a	Mehr als eine Geometrie pro Tabelle	Struktur 1:1 in DB übernehmen	CLASS A = Attribut_1; Geom_1: Line; Geom_2: Surface; END A;	CREATE TABLE A (ID PK, Attribut_1, Geom_1 geometry(Linestring), Geom_2 geometry(Polygon))	Es hängt von der Technologie ab. Z.B.: • PostGIS: diese Konstruktion ist möglich, aber mit gängiger GIS-Software nicht zu benutzen. • ESRI GDB: wird nicht unterstützt Man muss auch die evtl. begrenzten Möglichkeiten der Applikation beachten.
17.b	Mehr als eine Geometrie pro Tabelle	zusätzliche Geometrie-Attribute je in einer eigenen Tabelle	CLASS A = Attribut_1; Geom_1: Line; Geom_2: Surface; Attribut_2; END A;	CREATE TABLE A (ID PK, Attribut_1, Geom_1 geometry(Linestring), Attribut_2,) CREATE TABLE A_geom_2 (ID PK, Geom_2 geometry(Polygon))	
17.c	Mehr als eine Geometrie pro Tabelle	Tabelle bei zusätzlichen Geometrie-Attributen aufspalten	CLASS A = Attribut_1; Geom_1: Line; Geom_2: Surface; Attribut_2; END A;	CREATE TABLE A (ID PK, Attribut_1, Geom_1 geometry(Linestring),) CREATE TABLE A_geom_2 (ID PK, Geom_2 geometry(Polygon) Attribut_2,)	
17.d	Mehr als eine Geometrie pro Tabelle	pro Geometrie-Attribut ein redundanter Record bei dem nur der Geometrie-Wert	CLASS A = Attribut_1; Geom_1: Surface;	CREATE TABLE A (ID PK, xtf_id,	Aus einem Objekt werden zwei Records mit den selben Werten für xtf_id, Attribut 1 und Attribut 2. Der erste Record enthält als geom

		unterschiedlich ist	Geom_2: Line; Attribut_2; END A;	Attribut_1, Geom, Attribut_2,)	den Wert von Geom_1; der zweite Record enthält als geom den Wert von Geom_2. Nachteil: Typ der Spalte Geom kann unterschiedliche Arten enthalten Punkte und/oder Linien und/oder Flächen. Nachteil: Die Applikation bzw. der Nutzer muss sicherstellen, dass die redundanten Werte für xtf_id, Attribut 1 und Attribut 2 in den verschiedenen Records für das selbe Objekt identisch sind.
18	Kreisbögen	Kreisbögen nicht in DB unterstützt			Viele DBs unterstützen Kreisbögen. Wenn Kreisbögen (durch die DB oder die GIS-Applikation) nicht unterstützt werden, muss die Geometrie verändert werden. Um in diesem Fall Datenverlust zu vermeiden, muss die Geometrie zusätzlich in einer eigenen Datenstruktur (z.B. als BLOB) gespeichert (und nachgeführt) werden.
19	Überlappungen	Umwandlung in eine ISO/OGC (simple feature) konforme Geometrie	A EXTENDS Surface = SURFACE WITHOUT OVERLAPS > 0.002;		Die Geometrie muss verändert werden da nicht kompatibel mit internationalen Standards. Um Datenverlust zu vermeiden muss die Geometrie zusätzlich in einer eigenen Datenstruktur (z.B. als BLOB) gespeichert (und nachgeführt) werden.
20	Überlappungen	Bedingung: zwei Flächen können sich nicht mehr als 0.002 Meter überschneiden	A EXTENDS Surface = SURFACE WITHOUT OVERLAPS > 0.002;		Kompliziert zu integrieren : • SQL based Datenbank: mit Trigger • ESRI GDB: nur Überprüfung Solche Einschränkungen sind auch über den Importprozess abfangbar.
21	Einschränkung	Ein Attribut einer Klasse von einem anderen abhängig	CLASS A = Attribut_1; Attribut_2 ;		Kompliziert zu integrieren : • SQL basierte Datenbank: mit Trigger • ESRI GDB: nur Überprüfung

			MANDATORY CONSTRAINT Attribut_1 <= Attribut_2; END A;		Solche Einschränkung wären auch über Importprozesse abfangbar.
22	Genauigkeit	INTERLIS gibt die Genauigkeit der Koordinaten durch Nachkommastellen (z. B. hier 1 mm)	COORD 480000.000 .. 840000.000 [m], 7000.000 .. 300000.000 [m], 200.000 .. 5000.000 [m];	• ESRI GDB: Resolution : 1mm Tolerance (ESRI Empfehlung): mindestens den doppelten Wert der Resolution: 2mm	• PostGreSQL: Koordinaten werden mit der vollen Auflösung (Nachkommastellen) abgespeichert. • ESRI GDB: kennen das Konzept von Tolerance und Resolution, d.h. dass Koordinaten auf ein virtuelles Gitternetz mit einer gewissen Distanz zwischen den Maschen „gesnappt“ werden. Die Resolution darf nicht zu grob definiert werden.
23	Datentyp OID	Ein Attributtyp wird als UUID OID Wertebereich definiert.	CLASS A = Attribut_1 : MANDATORY INTERLIS.UUIDOID; END A;		• PostGreSQL: GUID • ESRI GDB: GUID • Oracle: RAW(16)

3.5 CHBase

PK = primary key
FK = foreign key

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
24	chbase: LocalisedText	Ein Attributtyp wird als LocalisedText definiert.	STRUCTURE LocalisedText = Language: LanguageCode_ISO639_1; Text: MANDATORY TEXT; END LocalisedText; CLASS A = Attribut_1 : ….LocalisedText; END A;	CREATE TABLE A (ID PK, Attribut_1, Attribut_1_Language)	Da nur eine Sprache unterstützt ist, kann man in der Haupttabelle ein Attribut einfügen.
25.a	chbase:MultilingualText	Attribut als eigene Tabelle	STRUCTURE LocalisedText = Language: LanguageCode_ISO639_1; Text: MANDATORY TEXT; END LocalisedText; STRUCTURE MultilingualText = LocalisedText : BAG {1..*} OF LocalisedText; UNIQUE (LOCAL) LocalisedText:Language; END MultilingualText; CLASS A = Attribut 1 : LocalisationCH_V1. MultilingualText; END A;	CREATE TABLE A_Attribut1 (ID PK, A_Attribut1, Language, Text CONSTRAINT A_Attribut1_FK FOREIGN KEY (A_Attribut1) REFERENCES A(ID) MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION,) CREATE TABLE A (ID PK)	Die Struktur MultilingualText ist als eine Tabelle modelliert, die einen Fremdschlüssel enthält, um die Verbindung mit der Haupttabelle A zu machen.

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
25.b	chbase:MultilingualText	In Haupttabelle einbetten	STRUCTURE LocalisedText = Language: LanguageCode_ISO639_1; Text: MANDATORY TEXT; END LocalisedText; STRUCTURE MultilingualText = LocalisedText : BAG {1..*} OF LocalisedText; UNIQUE (LOCAL) LocalisedText:Language; END MultilingualText; CLASS A = Attribut 1 : LocalisationCH_V1. MultilingualText; END A;	CREATE TABLE A (ID PK, Attribut1_DE, Attribut1_FR, Attribut1_IT)	Möglich, aber kann unangenehm sein, da die Haupttabelle viele Attribute enthält.
26	chbase: CatalogueReference	Ein Attributtyp wird als CatalogueReference definiert.	CLASS X EXTENDS …Catalogues.Item = Code : MANDATORY TEXT; Description : MANDATORY TEXT; END X; STRUCTURE Y EXTENDS ...Catalogues.CatalogueReference = Reference (EXTENDED) : REFERENCE TO X; END Y; CLASS A = Attribut_1 : Y;	CREATE TABLE X (ID PK, Code, Description) CREATE TABLE A (ID PK, Attribut_1 CONSTRAINTAttribut_1_FK FOREIGN KEY (Attribut_1) REFERENCES X(Code) MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION,)	• SQL based Datenbank: die Tabelle X enthält alle möglichen Werte des Katalogs. Dann wird die Verbindung mit der Haupttabelle mit einem Fremdschlüssel gemacht. • ESRI GDB: CodedValueDomains

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
			END A;		
27.a	chbase: Item		CLASS X EXTENDS …Catalogues.Item = Code : MANDATORY TEXT; Description : MANDATORY TEXT; END X;	CREATE TABLE Item (ID PK, Class, // z.B. ‚X‘ Code, Description)	Eine Codetabelle für alle Klassen die Item erweitern. Erleichtert/Ermöglich evtl. Nutzung von bestehender Software-Library.
27.b	chbase: Item „flachwalzen“	Ein Attributtyp wird als CatalogueReference definiert.	CLASS X EXTENDS …Catalogues.Item = Code : MANDATORY TEXT; Description : MANDATORY TEXT; END X; STRUCTURE Y EXTENDS ...Catalogues.CatalogueReference = Reference (EXTENDED) : REFERENCE TO X; END Y; CLASS A = Attribut_1 : Y; END A;	CREATE TABLE A (ID PK, Attribut_1_Code, Attribut_1_Description)	Starke Vereinfachung für Auswertungen, relativ mühsam, wenn man die Codetabelle ändern will
28	chbase: MultLine	Ein Geometrietyp wird als MultiLine definiert.	Line = POLYLINE WITH (STRAIGHTS, ARCS) VERTEX Coord2; STRUCTURE LineStructure = Line: Line; END LineStructure; STRUCTURE MultiLine = Lines: BAG {1..*} OF LineStructure;	CREATE TABLE X (ID PK, Geo_Obj geometry MultiLinestring,)	Hängt von der Technologie ab : • PostGIS: MULTIPart unterstützt • ESRI GDB: MulitPart unterstützt

Nummer	Konstrukt Use Case	Beschreibung	Beispiel	Vereinfachung (Bsp SQL)	Kommentare
			END MultiLine; CLASS X = Geo_Obj : MANDATORY MultiLine; END X;		
29	chbase: MultiSurface	Ein Geometrietyp wird als MultiSurface definiert.	Surface = SURFACE WITH (STRAIGHTS, ARCS) VERTEX Coord2; STRUCTURE SurfaceStructure = Surface: Surface; END SurfaceStructure; STRUCTURE MultiSurface = Surfaces: BAG {1..*} OF SurfaceStructure; END MultiSurface; CLASS X = Geo_Obj : MANDATORY MultiSurface; END X;	CREATE TABLE X (ID PK, Geo_Obj geometry MultiPolygon,)	Hängt von der Technologie ab : • PostGIS: MultiPart unterstützt • ESRI GDB: MultiPart unterstützt

4 Anhang

4.1 „Zahlen“ und „Text“ Datentypen: INTERLIS/PostGreSQL/ESRI GDB

	INTERLIS	PostGreSQL		ESRI GDB	
Zahlen	Wertebereiche: Z.B. 0 .. 130	Smallint	-32768 bis +32767	Short integer	-32,768 bis 32,767
		Integer	-2147483648 bis +2147483647	Long integer	-2,147,483,648 bis 2,147,483,647
		Bigint	-9223372036854775808 to 9223372036854775807		
	Wertebereiche: z.B. 0.0 .. 130.0	decimal	no limit	Single-precision floating-point number (float)	Ca. -3.4E38 bis 1.2E38
		numeric	no limit	Double-precision floating-point number (double)	Ca. -2.2E308 bis 1.8E308
		real	6 decimal digits precision		
		double precision	15 decimal digits precision		
Text	Mit Längendefinition: TEXT*30	character varying(n), var-char(n)	variable-length with limit	Text(Länge)	
		character(n), char(n)	fixed-length, blank padded		
		text	variable unlimited length		