ili2db-Anleitung

Überblick

Ili2pg bzw. ili2gpkg ist ein in Java erstelltes Programm, das eine Interlis-Transferdatei (itf oder xtf) einem Interlis-Modell entsprechend (ili) mittels 1:1-Transfer in eine Datenbank (PostgreSQL/Postgis bzw. GeoPackage) schreibt oder aus der Datenbank mittels einem 1:1-Transfer eine solche Transferdatei erstellt. Folgende Funktionen sind möglich:

- 1:1-Umwandlung einer Modelldatei in ein Datenbankschema.
- 1:1-Import einer beliebigen Transferdatei mit dazugehöriger Modelldatei in eine Datenbank.
- 1:1-Export von Datenbanktabellen in eine Interlis-Transferdatei.
- 1:1-Export von Datenbanktabellen in eine GML-Transferdatei[1].

1:1-Import in die Datenbank

Der 1:1-Import schreibt alle Objekte (im Sinne der eigentlichen Daten) der Interlis-Transferdatei in die Datenbank. Falls die Tabellen in der Datenbank resp. im Schema noch nicht existieren, werden die Tabellen und falls nötig das Schema beim Import angelegt.
Es besteht die Möglichkeit ein Schema mit leeren Tabellen aus dem Interlis-Modell in der Datenbank zu erstellen (nur PostGIS).
Area- und Surface-Geometrien können bei Interlis 1 optional polygoniert werden.
Kreisbögen werden als Kreisbögen importiert und somit nicht segmentiert oder können optional auch segmentiert werden.

Attribute vom Interlis-Datentyp „Enumeration“ können wahlweise auch zusätzlich als Text importiert werden (z.B. BB-Art 0 = „Gebaeude“).

Den Geometrien kann mittels Parameter ein EPSG-Code zugewiesen werden. Die Geometrie-Attribute können optional indexiert werden.

1:1-Export aus der Datenbank

Der 1:1-Export schreibt alle Tabellen eines Interlis-Modells in eine Interlis-Transferdatei.
Geometrien vom Typ Area und Surface werden bei Interlis 1 während dem Export in Linien umgewandelt.

Laufzeitanforderungen

Das Programm setzt Java 1.6 voraus.
PostGIS: Als Datenbank muss mindestens PostgreSQL 8.3 und PostGIS 1.5 vorhanden sein.

Lizenz

GNU Lesser General Public License

Funktionsweise

In den folgenden Abschnitten wird die Funktionsweise anhand einzelner Anwendungsfälle beispielhaft beschrieben. Die detaillierte Beschreibung einzelner Funktionen ist im Kapitel „Referenz“ zu finden.

Import-Funktionen

Fall 1

Die Tabellen existieren nicht und sollen in der Datenbank angelegt werden.

PostGIS: `java -jar ili2pg.jar --schemaimport --dbdatabase mogis --dbusr julia -dbpwd romeo path/to/dm01av.ili`
GeoPackage: `java -jar ili2gpkg.jar --schemaimport --dbfile mogis.gpkg`

[1] GML 3.2; die verwendeten Kodierungsregeln entsprechen eCH-0118-1.0
[2] Der SQL-Name ergibt sich aus den Namenskonventionen. Die konkrete Übersetzung ist in der

```
path/to/dm01av.ili
```

Es werden keine Daten importiert, sondern nur die leeren Tabellen angelegt.
PostGIS: Die leeren Tabellen werden im Default-Schema des Benutzers julia angelegt. Die Geometrie-Spalten werden in der Tabelle public.geometry_columns registriert.
Als Host wird der lokale Rechner angenommen und für die Verbindung zur Datenbank der Standard-Port.
GeoPackage: Die Geometrie-Spalten werden in den Tabellen gpkg_contents und gpkg_geometry_columns registriert.
Falls die Datei mogis.gpkg noch nicht existiert, wird sie erzeugt und mit den für GeoPackage nötigen Metatabellen initialisiert.

Fall 2 (nur PostGIS)

Das gewünschte Schema und die Tabellen existieren nicht und es soll das DB-Schema und -Datenmodell angelegt werden:

PostGIS: `java -jar ili2pg.jar --schemaimport --dbdatabase mogis --dbschema dm01av --dbusr julia -dbpwd romeo path/to/dm01av.ili`

Es werden keine Daten importiert, sondern nur das Schema dm01av und die leeren Tabellen angelegt. Die Geometrie-Spalten werden in der Tabelle public.geometry_columns registriert.

Fall 3

Die Tabellen existieren nicht und sollen in der Datenbank angelegt werden und die Daten sollen importiert werden:

PostGIS: `java -jar ili2pg.jar --import --dbhost ofaioi4531 --dbport 5432 --dbdatabase mogis --dbusr julia --dbpwd romeo -log path/to/logfile path/to/260100.itf`
GeoPackage: `java -jar ili2gpkg.jar --import --dbfile mogis.gpkg -log path/to/logfile path/to/260100.itf`

Alle Tabellen werden in der Datenbank erstellt und das Itf 260100.itf importiert. Die Geometrie-Spalten werden registriert. Als Primary-Key wird ein zusätzliches Attribut erstellt (t_id). Zusätzlich wir ein t_basket Attribut erstellt. Dieses zeigt als Fremdschlüssel auf eine Meta-Hilfstabelle (Importdatum, Benutzer, Modellname, Pfad der Itf-Datei).
Die Aufzähltypen werden in Lookup-Tables abgebildet.
Es wird ein Logfile angelegt. Dieses enthält Zeitpunkt des Imports, Name des Benutzers, Datenbankparameter (ohne Passwort), Name (ganzer Pfade) der Ili- und Itf-Datei, sämtliche Namen der importierten Tabellen inkl. Anzahl der importierten Elemente pro Tabelle. Allfällige Fehlermeldungen (bei Importabbruch) werden auch in die Logdatei geschrieben.

Fall 4

Die Tabellen existieren bereits und der Inhalt der Tabellen soll erweitert werden:

PostGIS: `java -jar ili2pg.jar --import --dbdatabase mogis --dbusr julia --dbpwd romeo path/to/260100.itf`
GeoPackage: `java -jar ili2gpkg.jar --import --dbfile mogis.gpkg path/to/260100.itf`

Das Itf 260100.itf wird importiert und die Daten den bereits vorhanden Tabellen hinzugefügt. Die Tabellen können zusätzliche Attribute enthalten (z.B. bfsnr, datum etc.), welche beim Import leer bleiben.

Fall 5

Die Tabellen existieren bereits und der Inhalt der Tabellen soll durch den Inhalt des itf ersetzt werden:

PostGIS: `java -jar ili2pg.jar --import --deleteData --dbdatabase mogis --dbusr julia --dbpwd romeo path/to/260100.itf`
GeoPackage: `java -jar ili2gpkg.jar --import --deleteData --dbfile mogis.gpkg path/to/260100.itf`

Das Itf 260100.itf wird importiert und die bestehenden Daten in den bereits vorhanden Tabellen

gelöscht. Die Tabellen können zusätzliche Attribute enthalten (z.B. bfsnr, datum etc.), welche beim Import leer bleiben.

Fall 6

Enumerations werden zusätzlich als Textattribut hinzugefügt:

PostGIS: `java -jar ili2pg.jar --import --createEnumTxtCol --dbdatabase mogis --dbusr julia --dbpwd romeo path/to/260100.itf`
GeoPackage: `java -jar ili2gpkg.jar --import --createEnumTxtCol --dbfile mogis.gpkg path/to/260100.itf`

Das Itf wird in die Datenbank importiert. Zusätzlich werden die Attribute vom Typ Enumeration in ihrer Textrepräsentation (Attribut „art“ = 0 ⇒ „art_txt“ = „Gebaeude“) hinzugefügt.

Fall 7

Den Geometrien wird ein spezieller SRS (Spatial Reference System) Identifikator hinzugefügt:

PostGIS: `java -jar ili2pg.jar --import --defaultSrsAuth epsg --defaultSrsCode 2056 --dbdatabase mogis --dbusr julia --dbpwd romeo path/to/260100.itf`
GeoPackage: `java -jar ili2gpkg.jar --import --defaultSrsAuth epsg --defaultSrsCode 2056 --dbfile mogis.gpkg path/to/260100.itf`

Das Itf wird in die Datenbank importiert. Zusätzlich wird jeder Geometrie eine SRS-ID (EPSG-Code 2056) hinzugefügt. Ebenfalls wird derselbe Identifikator für die Registrierung der Geometriespalten in den Metatabellen der Datenbank benutzt.

Fall 8

Geometrien werden indexiert:

PostGIS: `java -jar ili2pg.jar --import --createGeomIdx --dbdatabase mogis --dbusr julia --dbpwd romeo path/to/260100.itf`
GeoPackage: `java -jar ili2gpkg.jar --import --createGeomIdx --dbfile mogis.gpkg path/to/260100.itf`

Das Itf wird in die Datenbank importiert. Die Geometrien werden indexiert.

Fall 9

Tauchen beim Import des Itf Fehler auf (z. B. mangelnde Modellkonformität oder verletzte Constraints in der DB), bricht der Import ab und keine Daten werden importiert. D.h. der Import in die Datenbank ist ein einzelner Commit.

Export-Funktionen

Fall 1

Die Tabellen werden aus der Datenbank in eine Interlis 1-Transfer-Datei geschrieben:

PostGIS: `java -jar ili2pg.jar --export --models DM01AV --dbhost ofaioi4531 --dbport 5432 --dbdatabase mogis --dbusr julia --dbpwd romeo path/to/output.itf`
GeoPackage: `java -jar ili2gpkg.jar --export --models DM01AV --dbfile mogis.gpkg path/to/output.itf`

Die Tabellen werden dem Interlis-Modell DM01AV entsprechend in die Interlis 1-Transferdatei output.itf geschrieben. Fehlende Tabellen in der Datenbank werden dementsprechend als leere Tabellen oder gar nicht (gemäss Definition im Datenmodell) in die Datei geschrieben. Fehlende Attribute in einer Datenbanktabelle werden mit einem „@“ substituiert.

Anhand des Parameters --models wird definiert, welche Daten exportiert werden. Alternativ kann auch der Parameter --topics oder --baskets verwendet werden, um die zu exportierenden Daten auszuwählen. Einer dieser Parameter muss also zwingend beim Export angegeben werden.

Fall 2

Die Tabellen werden aus der Datenbank in eine Interlis 2-Transfer-Datei geschrieben:

PostGIS: `java -jar ili2pg.jar --export --models DM01AV --dbhost ofaioi4531 --dbport 5432 --dbdatabase mogis --dbusr julia --dbpwd romeo path/to/output.xtf`
GeoPackage: `java -jar ili2gpkg.jar --export --models DM01AV --dbfile mogis.gpkg path/to/output.xtf`

Die Tabellen werden dem Interlis-Modell DM01AV entsprechend in das die Interlis 2-Transferdatei output.xtf geschrieben. Fehlende Tabellen und Attribute in der Datenbank werden gar nicht in die Datei geschrieben.
Anhand des Parameters --models wird definiert, welche Daten exportiert werden. Alternativ kann auch der Parameter --topics oder --baskets verwendet werden, um die zu exportierenden Daten auszuwählen. Einer dieser Parameter muss also zwingend beim Export angegeben werden.

Referenz

In den folgenden Abschnitten werden einzelne Aspekte detailliert, aber isoliert, beschrieben. Die Funktionsweise als Ganzes wird anhand einzelner Anwendungsfälle beispielhaft im Kapitel „Funktionsweise“ (weiter oben) beschrieben.

Aufruf-Syntax

PostGIS: `java -jar ili2pg.jar [Options] [file]`
GeoPackage: `java -jar ili2gpkg.jar [Options] [file]`

Optionen:

`--import`	Importiert Daten aus einer Transferdatei in die Datenbank. Die Tabellen werden implizit auch angelegt, falls sie noch nicht vorhanden sind (siehe Kapitel Abbildungsregeln). Die Tabellen können zusätzliche Attribute enthalten (z.B. bfsnr, datum etc.), welche beim Import leer bleiben. TODO Die Tabellen sind schon vorhanden (und entsprechen (nicht) der ili-Klasse)
`--update`	Aktualisiert die Daten in der Datenbank anhand einer Transferdatei, d.h. neue Objekte werden eingefügt, bestehende Objekte werden aktualisiert und in der Transferdatei nicht mehr vorhandene Objekte werden gelöscht. Diese Funktion bedingt, dass das Datenbankschema mit der Option --createBasketCol erstellt wurde, und dass die Klassen und Topics eine stabile OID haben.
`--export`	Exportiert Daten aus der Datenbank in eine Transferdatei. Mit dem Parameter --models, --topics oder --baskets wird definiert, welche Daten exportiert werden. Ob die Daten im Interlis 1-, Interlis 2- oder GML-Format geschrieben werden, ergibt sich aus der Dateinamenserweiterung der Ausgabedatei. Für eine Interlis 1-Transferdatei muss die Erweiterung .itf verwendet werden. Für eine GML-Transferdatei muss die Erweiterung .gml verwendet werden. Die Optionen --topics und --baskets bedingen, dass das Datenbankschema mit der Option --createBasketCol erstellt wurde.
`--schemaimport`	Erstellt die Tabellenstruktur in der Datenbank (siehe Kapitel Abbildungsregeln).
`--dbhost host`	**PostGIS:** Der hostname der Datenbank. Default ist localhost.
`--dbport port`	**PostGIS:** Die Port-Nummer, unter der die Datenbank angesprochen warden kann. Default ist 5432.
`--dbdatabase database`	**PostGIS:** Der Name der Datenbank.

`--dbusr username`	**PostGIS:** Der Benutzername für den Datenbankzugang und Einträge in Metatabellen. **GeoPackage:** Der Benutzername für Einträge in Metatabellen.
`--dbpwd password`	**PostGIS:** Das Passwort für den Datenbankzugriff.
`--dbschema schema`	**PostGIS:** Definiert den Namen des Datenbank-Schemas. Default ist kein Wert, d.h. das aktuelle Schema des Benutzers der mit –user definiert wird.
`--dbfile filename`	**GeoPackage:** Name der GeoPackage-Datei.
`--deleteData`	bei einem Datenimport (--import) werden alle Daten in den existierenden/benutzten Tabellen gelöscht (Mit DELETE, die Tabellenstruktur bleibt unverändert).
`--defaultSrsAuth auth`	SRS Authority für Geometriespalten, wo sich dieser Wert nicht ermitteln lässt (für ili1 und ili2.3 immer der Fall). Default ist EPSG
`--defaultSrsCode code`	SRS Code für Geometriespalten, wo sich dieser Wert nicht ermitteln lässt (für ili1 und ili2.3 immer der Fall). Default ist 21781
`--modeldir path`	Dateipfade, die Modell-Dateien (ili-Dateien) enthalten. Mehrere Pfade können durch Semikolon „;“ getrennt werden. Es sind auch URLs von Modell-Repositories möglich. Default ist %ILI_FROM_DB;%XTF_DIR;http://models.interlis.ch/;%JAR_DIR %ILI_FROM_DB ist ein Platzhalter für die in der Datenbank vorhandenen Modelle (in der Tabelle t_ili2db_model). %XTF_DIR ist ein Platzhalter für das Verzeichnis mit der Transferdatei. %JAR_DIR ist ein Platzhalter für das Verzeichnis des ili2db Programms (ili2pg.jar bzw. ili2gpkg.jar Datei). Der erste Modellname (Hauptmodell), zu dem ili2db die ili-Datei sucht, ist nicht von der INTERLIS-Sprachversion abhängig. Es wird in folgender Reihenfolge nach einer ili-Datei gesucht: zuerst INTERLIS 2.3, dann 1.0 und zuletzt 2.2. Beim Auflösen eines IMPORTs wird die INTERLIS Sprachversion des Hauptmodells berücksichtigt, so dass also z.B. das Modell Units für ili2.2 oder ili2.3 unterschieden wird.
`--models modelname`	Namen des Modells (nicht zwingend identisch mit dem Dateinamen!), für das die Tabellenstruktur in der Datenbank erstellt werden soll. Mehrere Modellnamen können durch Semikolon „;“ getrennt werden. Normalerweise muss der Namen nicht angegeben werden, und das Programm ermittelt den Wert automatisch aus den Daten. Wird beim --schemaimport nur eine ili-Datei als file angegeben, wird der Name des letzten Modells aus dieser ili-Datei als modelname genommen.
`--baskets BID`	BID der Baskets, die exportiert werden sollen. Mehrere BIDs können durch Semikolon „;“ getrennt werden.
`--topics topicname`	Topic-Namen der Baskets, die exportiert werden sollen. Mehrere Namen können durch Semikolon „;“ getrennt werden. Falls der Topic-Name in verschiedenen Modellen vorkommt, muss der qualifizierte Topic-Name verwendet werden.
`--createscript filename`	Erstellt zusätzlich zur Tabellenstruktur in der Datenbank ein SQL-Skript um die Tabellenstruktur unabhängig vom Programm erstellen zu können. Das Skript wird zusätzlich zu den Tabellen in der Datenbank erzeugt, d.h. es ist nicht möglich, nur das Skript zu erstellen (ohne Datenbank).
`--dropscript filename`	Erstellt ein SQL-Skript um die Tabellenstruktur unabhängig vom Programm löschen zu können.
`--noSmartMapping`	Alle strukturellen Abbildungsoptimierungen werden ausgeschaltet. (s.a. --smartInheritance, --coalesceCatalogueRef, --coalesceMultiSurface, --expandMultilingual)

`--smartInheritance`	Bildet die Vererbungshierarchie mit einer dymamischen Strategie ab. Für Klassen, die referenziert werden und deren Basisklassen nicht mit einer NewClass-Strategie abgebildet werden, wird die NewClass-Strategie verwendet. Abstrakte Klassen werden mit einer SubClass-Strategie abgebildet. Konkrete Klassen, ohne Basisklasse oder deren direkte Basisklassen mit einer SubClass-Strategie abgebildet werden, werden mit einer NewClass-Strategie abgebildet. Alle anderen Klassen werden mit einer SuperClass-Strategie abgebildet.
`--coalesceCatalogueRef`	Strukturattribute deren maximale Kardinalität 1 ist, deren Basistyp CHBase:CatalogueReference oder CHBase:MandatoryCatalogueReference ist und die ausser „Reference“ keine weiteren Attribute haben, werden direkt mit einem Fremdschlüssel auf die Ziel-Tabelle (die die konkrete CHBase:Item Klasse realisiert) abgebildet, d.h. kein Record in der Tabelle für die Struktur mit dem „Reference“ Attribut.
`--coalesceMultiSurface`	Strukturattribute deren maximale Kardinalität 1 ist, deren Basistyp CHBase:MultiSurface ist und die ausser „Surfaces“ keine weiteren Attribute haben, werden direkt als Spalte mit dem Typ MULTISURFACE (oder MULTIPOLYGON, falls --strokeArcs) abgebildet.
`--expandMultilingual`	Strukturattribute deren maximale Kardinalität 1 ist, deren Basistyp LocalisationCH_V1.MultilingualText oder LocalisationCH_V1.MultilingualMText ist und die ausser „LocalisedText“ keine weiteren Attribute haben, werden direkt als Spalten in der Tabelle des Strukturattributes abgebildet, d.h. keine Records in den Tabellen für die Multilingual-Strukturen.
`--createGeomIdx`	Erstellt für jede Geometriespalte in der Datenbank einen räumlichen Index. (siehe Kapitel Abbildungsregeln/Geometrieattribute)
`--createEnumColAsItfCode`	Bildet bei Aufzählungsattributen den Aufzählungswert als ITF-Code ab. Diese Option ist nur zulässig, wenn im Modell keine Erweiterungen von Aufzählungen vorkommen. Ohne diese Option wird der XTF-Code als Aufzählwert in der Datenbank verwendet. (siehe Kapitel Abbildungsregeln/Aufzählungen)
`--createEnumTxtCol`	Erstellt für Aufzählungsattribute eine zusätzliche Spalte mit dem Namen des Aufzählwertes. (siehe Kapitel Abbildungsregeln/Aufzählungen)
`--createEnumTabs`	Erstellt pro Aufzählungsdefinition eine Tabelle mit den einzelnen Aufzählwerten. (siehe Kapitel Abbildungsregeln/Aufzählungen)
`--createSingleEnumTab`	Erstellt eine einzige Tablle mit allen Aufzählwerten aller Aufzählungsdefinitionen. (siehe Kapitel Abbildungsregeln/Aufzählungen)
`--createStdCols`	Erstellt in jeder Tabelle zusätzliche Metadatenspalten T_User, T_CreateDate, T_LastChange. (siehe Kapitel Abbildungsregeln/Tabellen)
`--t_id_Name name`	Definiert den Namen für die interne technische Schlüsselspalte in jeder Tabelle (nicht zu verwechseln mit dem externen Transferidentifikator). Default ist T_Id. (siehe Kapitel Abbildungsregeln/Tabellen)
`--createTypeDiscriminator`	Erstellt für jede Tabelle (auch wenn das Modell keine Vererbung benutzt) eine Spalte für den Typdiskriminator. Für Klassen mit Vererbung wird die Spalte immer erstellt. (siehe Kapitel Abbildungsregeln/Tabellen)
`--structWithGenericRef`	Erstellt generische Spalten für den Fremdschlüssel bei Tabellen die Interlis-Strukturen abbilden. Ohne diese Option wird pro Strukturattribut eine Spalte erstellt (in der Tabelle, die die Struktur abbildet). (siehe Kapitel Abbildungsregeln/Strukturen)

`--disableNameOptimization`	Schaltet die Nutzung von unqualifizierten Klassennamen aus. Für alle Tabellennamen werden qualifizierte Interlis-Klassennamen (Model.Topic.Class) verwendet (und in einen gültigen Tabellennamen abgebildet). (siehe Kapitel Abbildungsregeln/Namenskonventionen)
`--nameByTopic`	Für alle Tabellennamen werden teilweise qualifizierte Interlis-Klassennamen (Topic.Class) verwendet (und in einen gültigen Tabellennamen abgebildet). (siehe Kapitel Abbildungsregeln/Namenskonventionen)
`--maxNameLength length`	Definiert die maximale Länge der Namen für Datenbankelemente (Tabellennamen, Spaltennamen , usw.) Default ist 60. Ist der Interlis-Name länger, wird er gekürzt. (siehe Kapitel Abbildungsregeln/Namenskonventionen)
`--sqlEnableNull`	Erstellt keine NOT NULL Anweisungen bei Spalten die Interlis-Attribute abbilden. (siehe Kapitel Abbildungsregeln/Attribute)
`--strokeArcs`	Segmentiert Kreisbogen beim Datenimport. Der Radius geht somit verloren. Die Kreisbogen werden so segmentiert, dass die Abweichung der erzeugten Geraden kleiner als die Koordinatengenauigkeit der Stützpunkte ist.
`--oneGeomPerTable`	**PostGIS:** Erzeugt Hilfstabellen, falls in einer Klasse/Tabelle mehr als ein Geometrie-Attribut ist, so dass pro Tabelle in der Datenbank nur eine Geometriespalte ist.
`--skipPolygonBuilding`	Bei ITF-Dateien werden die Linientabellen gelesen, so wie sie in der ITF-Datei sind, d.h. es werden keine Polygon gebildet.
`--skipPolygonBuildingErrors`	Bei ITF-Dateien werden aus den Linientabellen Polygone gebildet, aber Fehler werden ignoriert (aber trotzdem rapportiert).
`--keepAreaRef`	Bei ITF-Dateien wird für AREA Attribute der Gebietsreferenzpunkt als zusätzliche Spalte in der Tabelle eingefügt.
`--importTid`	Liest die Transferidentifikation (aus der Transferdatei) in eine zusätzliche Spalte T_Ili_Tid. (siehe Kapitel Abbildungsregeln/Tabellen)
`--createBasketCol`	Erstellt in jeder Tabelle eine zusätzlich Spalte T_basket um den Behälter identifizieren zu können. (siehe Kapitel Abbildungsregeln/Metadaten)
`--createFk`	Erzeugt eine Fremdschlüsselbedingung bei Spalten die Records in anderen Tabellen referenzieren.
`--createFkIdx`	Erstellt für jede Fremdschlüsselpalte in der Datenbank einen Index. Kann auch ohne die Option --createFk benutzt werden.
`--createUnique`	Erstellt für INTERLIS-UNIQUE-Constraints in der Datenbank UNIQUE Bedingungen (sofern abbildbar).
`--log filename`	Schreibt die log-Meldungen in eine Datei.
`--gui`	Startet ein einfaches GUI.
`--trace`	Erzeugt zusätzliche Log-Meldungen (wichtig für Programm-Fehleranalysen)
`--help`	Zeigt einen kurzen Hilfetext an.
`--version`	Zeigt die Version des Programmes an.

Abbildungsregeln

Klassen/Strukturen

Je nach Programmoption, werden Klassen unterschiedlich abgebildet. Die Abbildungsregeln für den Tabellennamen sind im Abschnitt Namenskonventionen beschrieben.

Nummer	Beispiel INTERLIS	Beispiel SQL	Kommentare
1	CLASS A = END A;	CREATE TABLE A (T_Id integer PRIMARY KEY);	Für jede Klasse wird eine Tabelle erstellt. Jede Tabelle hat mindestens eine Spalte T_Id. Diese Spalte ist der Datenbank interne

			Primärschlüssel (und nicht die TID aus der Transferdatei).
2	CLASS A = END A;	CREATE TABLE A (T_Id integer PRIMARY KEY, T_Type varchar(60) NOT NULL);	Mit der Option --createTypeDiscriminator erhält jede Tabelle (die eine Klasse oder Struktur repräsentiert, die keine Basisklasse hat) eine zusätzliche Spalte T_Type. Diese Spalte enthält den konkreten Klassenname (der SQL-Name des qualifizierten INTERLIS-Klassennamens[2]) des Objektes jedes einzelnen Records. Tabellen für Klassen die eine Basisklasse haben, erhalten diese Spalte nicht.
3	CLASS A = END A;	CREATE TABLE A (T_Id integer PRIMARY KEY, T_LastChange timestamp NOT NULL, T_CreateDate timestamp NOT NULL, T_User varchar(40) NOT NULL);	Mit der Option --createStdCols erhalten alle Tabellen drei zusätzliche Spalten für den Zeitpunkt der letzten Änderung, den Zeitpunkt der Erstellung und den Benutzer, der die letzte Änderung durchgeführt hat. Diese Spalten müssen durch die Applikation nachgeführt werden, und werden typischerweise für die Implementierung eines optimistischen Lockings benötigt/benutzt.
4	CLASS A = END A;	CREATE TABLE A (T_Id integer PRIMARY KEY, T_Ili_Tid varchar(200) NULL);	Mit der Option --importTid erhält jedes Tabelle (die eine Klasse repräsentiert, die keine Basisklasse hat) eine zusätzliche Spalte T_Ili_Tid. Diese Spalte enthält die TID aus der Transferdatei. Diese Spalte ist NICHT der Datenbank interne Primärschlüssel.
5	CLASS A = END A;	CREATE TABLE A (oidname integer PRIMARY KEY);	Mit der Option --t_id_Name oidname wird der Namen der Spalte für den Datenbank internen Primärschlüssel (nicht die Spalte für die TID aus der Transferdatei) festgelegt.
6	STRUCTURE C = END C;	CREATE TABLE C (T_Id integer PRIMARY KEY, T_seq integer NOT NULL);	Strukturen werden im Allgemeinen abgebildet wie Klassen. Die Strukturtabelle enthält zusätzlich eine Spalte T_seq, die die Reihenfolge der Strukturelement festlegt. Da Strukturelemente keine TID haben, erhalten sie auch mit der Option --importTid kein Spalte T_Ili_Tid.

Vererbung

Im allgemeinen lässt sich Vererbung nach drei unterschidlichen Strategien abbilden:
NewClass: Diese Strategie ist für jede Klasse möglich. Bei dieser Strategie wird für eine Klasse eine neue Tabelle angelegt, ein Interlis-Objekt verteilt sich somit auf Records in mehreren Tabellen.
SuperClass: Diese Strategie ist nur für Klassen mit einer Super-Klasse möglich. Bei dieser Strategie wird für die Klasse keine neue Tabelle angelegt, d.h. die Attribute der Klasse werden als weitere Spalten in der Tabelle der Super-Klasse ergänzt.
SubClass: Diese Strategie ist nur für Klassen mit mindestens einer Sub-Klasse möglich. Bei dieser

[2] Der SQL-Name ergibt sich aus den Namenskonventionen. Die konkrete Übersetzung ist in der Tabelle T_ILI2DB_CLASSNAME hinterlegt.

Strategie wird für eine Klasse keine neue Tabelle angelegt, d.h. die Attribute der Klasse werden als weitere Spalten in den Tabellen der Sub-Klassen ergänzt.

ili2db bildet die Vererbung nach einer je nach Klasse unterschiedlichen Strategie (`--smartInheritance`) oder für alle Klassen einheitlich nach der NewClass-Strategie (`--noSmartMapping`) ab.
Bei --smartInheritance wird wie folgt abgebildet: Fuer Klassen, die referenziert werden und deren Basisklassen nicht mit einer NewClass-Strategie abgebildet werden, wird die NewClass-Strategie verwendet. Abstrakte Klassen werden mit einer SubClass-Strategie abgebildet. Konkrete Klassen, ohne Basisklasse oder deren direkte Basisklassen mit einer SubClass-Strategie abgebildet werden, werden mit einer NewClass-Strategie abgebildet. Alle anderen Klassen werden mit einer SuperClass-Strategie abgebildet.

Nummer	Beispiel INTERLIS	Beispiel SQL	Kommentare

1	CLASS A =
	  Attribut_1 : TEXT*20;
	END A;
	
	CLASS B EXTENDS A =
	  Attribut_2 : TEST*20;
	END B;
		CREATE TABLE A (
		  T_Id integer PRIMARY KEY,
		  T_Type varchar(60) NOT NULL,
		  Attribut_1 varchar(20)
		);
		CREATE TABLE B (
		  T_Id integer PRIMARY KEY,
		  Attribut_2 varchar(20)
		);
			Für jede Klasse wird eine Tabelle erstellt. Ein Objekt A ergibt ein Record in Tabellen A. Ein Objekt B ergibt je ein Record in Tabellen A und B. Die T_Id ist bei beiden Records identisch.

			TODO Beispiele für --smartInheritance ergänzen

EXTENDED Attribute ergeben keine Spalte, nur die Basis-Definition des Attributs ergibt eine Spalte.

Attribute (allgemein)

TODO

Beziehungen/Referenzattribute

TODO

Geometrieattribute (allgemein)

TODO

SURFACE/AREA/ITF/XTF

TODO

Strukturattribute

Strukturen werden im Allgemeinen abgebildet wie Klassen.

Nummer	Beispiel INTERLIS	Beispiel SQL	Kommentare

1	STRUCTURE C =
	END C;
	
	CLASS D =
	  attr1 : LIST OF C;
	  attr2 : LIST OF C;
	END D;
		CREATE TABLE C (
		  T_Id integer PRIMARY KEY,
		  T_seq integer NOT NULL,
		  D_attr1 integer,
		  D_attr2 integer
		);
		CREATE TABLE D (
		  T_Id integer PRIMARY KEY
		);
			Für jedes Strukturattribut wird in der Tabelle der Struktur eine Spalte für den Fremdschlüssel erstellt. Der Name der Spalte ist der qualifizierte INTERLIS-Attributnamen[3]. Die Strukturtabelle enthält zusätzlich eine Spalte T_seq die die Reihenfolge der

[3] Der SQL-Name ergibt sich aus den Namenskonventionen. Die konkrete Übersetzung ist in der Tabelle T_ILI2DB_ATTRNAME hinterlegt.

			Strukturelement festlegt.
2	STRUCTURE C = END C; CLASS D = attr1 : LIST OF C; END D;	CREATE TABLE C (T_Id integer PRIMARY KEY, T_seq integer NOT NULL, T_ParentId integer NOT NULL T_ParentType varchar(60) NOT NULL T_ParentAttr varchar(60) NOT NULL); CREATE TABLE D (T_Id integer PRIMARY KEY);	Mit der Option --structWithGenericRef werden statt für jedes Strukturattribut eine Spalte nur drei Standardspalten T_ParentId, T_ParentType, T_ParentAttr angelegt. Diese drei Spalten bilden zusammen einen generischen Fremdschlüssel. T_ParentId ist die t_id des Objektes, das das Strukturelement enthält. T_ParentType ist die konkrete Klasse (der SQL-Name des qualifizierten INTERLIS-Klassennamens[4]) des Objektes, das das Strukturelement enthält. T_ParentAttr ist der Strukturattributname (der SQL-Name des unqualifizierten INTERLIS-Attributnamens) in der Klasse des Objektes, das das Strukturelement enthält.

Beispiel XML:

XTF-Datei

```
<BspTable.TopicA.D TID="2">
      <attr1>
            <BspTable.TopicA.C>
            </BspTable.TopicA.C>
            <BspTable.TopicA.C>
            </BspTable.TopicA.C>
      </attr1>
      <attr2>
            <BspTable.TopicA.C>
            </BspTable.TopicA.C>
      </attr2>
</BspTable.TopicA.D>
```

Beispiel für Abbildungsvariante 1:

Tabelle C

t_id	t_seq	D_attr1	D_attr2
7	0	6	
8	1	6	
9	0		6

Tabelle D

t_id	T_Ili_Tid
6	2

Beispiel für Abbildungsvariante 2:

Tabelle C

t_id	t_seq	t_parentid	t_parenttype	t_parentattr
7	0	6	D	attr1

[4] Der SQL-Name ergibt sich aus den Namenskonventionen. Die konkrete Übersetzung ist in der Tabelle T_ILI2DB_CLASSNAME hinterlegt.

8	1	6	D	attr1
9	0	6	D	attr2

Tabelle D	
t_id	T_Ili_Tid
6	2

Aufzählungen

TODO
Wie erfolgt die Nummerierung/Codierung?

Metadaten

Tabelle	Beschreibung
t_ili2db_attrname	Abbildung von Attributnamen
t_ili2db_basket	In der Datenbank vorhandene Baskets
t_ili2db_classname	Abbildung der qualifizierten Interlis Klassennamen auf Sql-Namen. Nicht aus jedem Eintrag gibt es eine Datenbank-Tabelle, je nach Abbildungsart der Interlis-Klasse wird der Sql-Name nur als Inhalt der Spalte t_type verwendet.
t_ili2db_dataset	In der Datenbank vorhandene Datensätze (Sammlung von Baskets)
t_ili2db_import	Statistik pro Importdatei
t_ili2db_import_basket	Statistik zu den importierten Daten pro Basket
t_ili2db_import_object	Statistik zu den importierten Daten pro Klasse
t_ili2db_inheritance	Abbildung der Interlis Klassen Vererbungshierarchie (in der Tabellen sind die qualifizierten Interlis Klassennamen)
t_ili2db_model	Modelle, die beim Import benötigt wurden (so dass der Export mit denselben Modellen erfolgen kann).
t_ili2db_settings	Programmeinstellungen für ili2db
t_ili2db_trafo	Konfiguration der semantischen Abbildung (insb. der Vererbung)
t_key_object	Hilfstabelle für den ID-Generator

TODO

Namenskonvention

Die Abbildung der Klassennamen in Tabellennamen erfolgt nach drei möglichen Strategien:
Unqualifiziert: Es wird der Klassennamen verwendet (ohne voranstellen von Topic- und/oder Model-Namen). Falls der Name schon benutzt ist, wird der voll qualifizierte Name verwendet.
Mit Topic qualifiziert: Dem unqualifizierten Klassennamen wird der Topic-Name vorangestellt. Falls der Name schon benutzt ist, wird der voll qualifizierte Name verwendet.
Voll qualifiziert: Der Tabellenname wird aus Model-, Topic- und Klassenname zusammengesetzt.
Falls der Tabellenname zu lang ist, wird er gekürzt, in dem die Vokale entfernt werden (ausser die ersten beiden und letzten beiden Buchstaben). Falls er danach immer noch zu lang ist, werden in der der Mitte des Namens Buchstaben entfernt.
Falls der Tabellenname nun einem SQL-Schlüsselwort entspricht, wird er um ein führeneds ‚a' ergänzt.
Falls der Tabellenname nun nicht eindeutig ist, wird er um eine Ziffer ergänzt: ‚0', ‚1', usw. bis er eindeutig ist.
Die automatische Namensabbildung kann übersteuert werden, indem vor dem ersten Import

entsprechende Einträge in der Tabelle t_ili2db_classname gemacht werden.