ili2pg-Anleitung

# Überblick

Ili2pg ist ein in Java erstelltes Programm, das eine Interlis-Transferdatei (itf oder xtf) einem Interlis-Modell entsprechend (ili) mittels 1:1-Transfer in eine PostgreSQL/Postgis-Datenbank 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 PostgreSQL/Postgis-Datenbankschema.
* 1:1-Import einer beliebigen Transferdatei mit dazugehöriger Modelldatei in eine PostgreSQL/Postgis-Datenbank.
* 1:1-Export von Datenbanktabellen in eine Interlis-Transferdatei.
* 1:1-Export von Datenbanktabellen in eine GML-Transferdatei[[1]](#footnote-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.

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.

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

Das gewünschte Schema und die Tabellen existieren nicht:

java -jar ili2pg.jar --import --dbhost ofaioi4531 --dbport 5432 --dbdatabase mogis --dbschema dm01av --dbusr julia --dbpwd romeo -log path/to/logfile path/to/260100.itf

Das Datenbank-Schema dm01av und alle Tabellen werden in der Datenbank mogis erstellt und das Itf 260100.itf importiert. Die Geometrie-Spalten werden in die Tabelle public.geometry\_columns 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 2

Die Tabellen existieren nicht und es soll nur das DB-Datenmodell (im Default Schema des Benutzers) angelegt werden:

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

Es werden keine Daten importiert, sondern nur die leeren Tabellen 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.

### Fall 3

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

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 4

Das Schema und die Tabellen existieren bereits und der Inhalt der Tabellen soll erweitert werden:

java -jar ili2pg.jar --import --dbdatabase mogis --dbschema dm01av --dbusr julia --dbpwd romeo 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

Das Schema und die Tabellen existieren bereits und der Inhalt der Tabellen soll durch den Inhalt des itf ersetzt werden:

java -jar ili2pg.jar --import --deleteData --dbdatabase mogis --dbschema dm01av --dbusr julia --dbpwd romeo 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:

java -jar ili2pg.jar --import --createEnumTxtCol --dbdatabase mogis --dbusr julia --dbpwd romeo path/to/260100.itf

Das Itf wird in das Default-Schema 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:

java -jar ili2pg.jar --import --defaultSrsAuth epsg --defaultSrsCode 2056 --dbdatabase mogis --dbusr julia --dbpwd romeo path/to/260100.itf

Das Itf wird in das Default-Schema importiert. Zusätzlich wird jeder Geometrie eine SRS-ID

(EPSG-Code 2056) hinzugefügt. Ebenfalls wird derselbe Identifikator für die Registrierung der Geometriespalten in die public.geometry\_columns Tabelle benutzt.

### Fall 8

Geometrien werden indexiert:

java -jar ili2pg.jar --import --createGeomIdx --dbdatabase mogis --dbusr julia --dbpwd romeo path/to/260100.itf

Das Itf wird in das Default-Schema 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 eines Schemas werden in eine Interlis 1-Transfer-Datei geschrieben:

java -jar ili2pg.jar --export --models DM01AV --dbhost ofaioi4531 --dbport 5432 --dbdatabase mogis --dbschema dm01av --dbusr julia --dbpwd romeo path/to/output.itf

Die Tabellen des Schemas dm01av 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.

ili2pg wählt anhand des Parameters --models welche Daten exportiert werden. Dieser muss also zwingend beim Export angegeben werden.

### Fall 2

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

java -jar ili2pg.jar --export --models DM01AV --dbhost ofaioi4531 --dbport 5432 --dbdatabase mogis --dbusr julia --dbpwd romeo path/to/output.xtf

Die Tabellen des default Schemas 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.

ili2pg wählt anhand des Parameters --models welche Daten exportiert werden. Dieser 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

java -jar ili2pg.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) |
| --export | Exportiert Daten aus der Datenbank in eine Transferdatei.  Mit dem Parameter --models 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. |
| --schemaimport | Erstellt die Tabellenstruktur in der Datenbank (siehe Kapitel Abbildungsregeln). |
| --dbhost host | Der hostname der Datenbank. Default ist localhost. |
| --dbport port | Die Port-Nummer, unter der die Datenbank angesprochen warden kann. Default ist 5432. |
| --dbdatabase database | Der Name der Datenbank. |
| --dbusr username | Der Benutzername für den Datenbankzugang. |
| --dbpwd password | Das Passwort für den Datenbankzugriff. |
| --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 ili2pg Programms (ili2pg.jar Datei). |
| --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. |
| --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. |
| --createGeomIdx | Erstellt für jede Geometriespalte in der Datenbank einen räumlichen Index. (siehe Kapitel Abbildungsregeln/Geometrieattribute) |
| --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. |
| --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) |
| --dbschema schema | Definiert den Namen des Datenbank-Schemas. Default ist kein Wert, d.h. das aktuelle Schema des Benutzers der mit –user definiert wird. |
| --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]](#footnote-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

ili2pg bildet die Vererbung nach der NewClass-Strategie ab, d.h. für jede Klasse eine Tabelle; ein Interlis-Objekt verteilt sich somit auf Records in mehreren Tabellen.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 A (  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.

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]](#footnote-3).  Die Strukturtabelle enthält zusätzlich eine Spalte T\_seq die die Reihenfolge der 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]](#footnote-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 |
| 8 | 1 | 6 | D | attr1 |
| 9 | 0 | 6 | D | attr2 |

|  |  |
| --- | --- |
| Tabelle D |  |
| t\_id | T\_Ili\_Tid |
| 6 | 2 |

### Aufzählungen

TODO

### Metadaten

|  |  |
| --- | --- |
| Tabelle | Beschreibung |
| t\_ili2db\_attrname | Abbildung von Attributnamen |
| t\_ili2db\_basket | In der Datenbank vorhandene Baskets |
| t\_ili2db\_classname | Abbildung der Klassennamen |
| 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 Vererbungshierarchie |
| t\_ili2db\_model | Modelle, die beim Import benötigt wurden (so dass der Export mit denselben Modellen erfolgen kann). |
| t\_ili2db\_settings | Einstellungen für ili2pg |
| t\_key\_object | Hilfstabelle für den ID-Generator |

TODO

### Namenskonvention

TODO

1. GML 3.2; die verwendeten Kodierungsregeln entsprechen einer Vorversion von eCH-0118 [↑](#footnote-ref-1)
2. Der SQL-Name ergibt sich aus den Namenskonventionen. Die konkrete Übersetzung ist in der Tabelle T\_ILI2DB\_CLASSNAME hinterlegt. [↑](#footnote-ref-2)
3. Der SQL-Name ergibt sich aus den Namenskonventionen. Die konkrete Übersetzung ist in der Tabelle T\_ILI2DB\_ATTRNAME hinterlegt. [↑](#footnote-ref-3)
4. Der SQL-Name ergibt sich aus den Namenskonventionen. Die konkrete Übersetzung ist in der Tabelle T\_ILI2DB\_CLASSNAME hinterlegt. [↑](#footnote-ref-4)