Template based relation database creator for mining industry

Jan Carlsson
Abstract

Template based relation database creator for mining industry

Jan Carlsson

The main parts of the production in the mining industry take place in different machines. These machines generate a big amount of data, which are being communicated, processed and stored for future processing. The mining industry goes through some changes in data processing. Today the main focus for the data processing is on the machines and the future goal is to redirect the main focus towards the mining technology. In the data processing this means standardization of the data processing so the data becomes uniform and easy to communicate between different processes.

This system makes long time storage available for mining companies that use machines producing data exported as XML documents described by an Iredes XML Schema. The system provides automatic translation of the XML Schema into a database schema, which can then be populated by data from the machines. The first problem was to find a system that could translate the template XML Schema required. The best translator found was XMLBeans that translates an XML Schema into Java classes. The developed system translates these classes into an SQL script which generates the schema for the database. The translation is automatic, requiring no manipulation of the template; just some settings in the external XMLBeans are required.
1 Introduction

The purpose of this work is to enable automatic translation of data produced by mining companies into relational databases. The developed system automatically generates relational database schemas for mining machines defined by some XML-Schema standards, such as the Iredes standard [1]. Iredes enables machines from different vendors to exchange data in a standardized way using the Iredes data format. The vendors do not have to invent new data formats for every new machine or model.

The developed system is called XsdDbBinder. It provides a standardized way for long time storage of machine data. Furthermore it simplifies exchange of data between different kinds of mining machines by a common representation in a relation database. It does not populate the database, which is future work.

The schema for the generated database is created from an existing XML schema (.XSD) file. The problem here is how to automatically translate an existing XML Schema, such as the Iredes template, into SQL scripts defining the corresponding relational database schema. As this Iredes template is a complex XML Schema we needed a general system for reading XML Schema documents. XMLBeans [4] was chosen for this. It is able to translate the entire Iredes template into Java classes. This system first uses XMLBeans to read and translate the XML Schema elements which it then converts into Java classes and methods. From the generated Java classes the system then inspects the return types to translate the classes to relational tables.

There are two general translations:

1. If a return type is a standard Java type it is translated into a table column. A standard Java type in this case means string, integer, float, Boolean.
2. Other return types with just a name become the table’s parent. In this case parent means parent in the XML tree structure. This means in a relation database the return type is set as the foreign key, and the foreign table is taken from the class name.

The system can modify an existing database, thus upgrading old versions of Iredes databases.
2 Background

2.1 Iredes

The International Rock Excavation Data Exchange Standard (IREDES) \cite{IREDES} was launched in 2000 by major players in the industry.

Iredes is a defined common language for easy and standardized data exchange between mining machines and central computer systems (Figure 1). As in other industries, this standard will have significant impact on the use of automated equipment. It shall provide easy ways of using multi-vendor machine installations in data controlled mining processes. At the same time it reduces cost for individual development of interfaces for all parties involved. The standard technical architecture is flexible and open - also for future applications. It uses modern technology enabling easy data exchange with commercial databases as well as open on line networking.

**Ground rules for the Iredes concept**

- The Iredes standard is built up for different kinds of machines. The machine components that are the same for different machines are reused in the same standard Iredes objects (for example general descriptions of diesel motor properties).
- Iredes provides flexibility without including basic data exchange details.
- The transferred information can be stored on diskettes as well as online machine status, using the same uniform data formats.
- Iredes provides independence between machines. It is easy to enlarge profiles to describe particular properties of a machine model.
- Iredes uses technologies that are established in the IT industry.

**Iredes Data Sets**

The Iredes data sets have four main parts: Equipment Profile, Application Profile, Administration Objects and Commonly used Objects, as illustrated by Figure 2-2.
The **equipment profile** defines equipment specific data. An equipment profile contains different data sets for different production areas, like *production performance* (DRPPerf), *production quality* (DRPQual), and *production plan* (DRPlan). From these equipment profiles the production areas might be derived from the application profile level. In the picture DRPPerf is derived from IRPPerfGenType.

- The **application profile** describes specific information used identically in all equipment profiles. The application profile data types are derived from the class IREDESType (Administration Objects).
- **Commonly used objects** are usable in many equipment profiles like *coordinate systems* and *diesel motors*.
- **Administration objects** are used for controlling the data. The class IREDESType describes the objects to administrate information for the organization of the Iredes data.

**Physical Iredes data sets**

The main parts of a physical Iredes data set are: *general header object, site header object, application object*, and *general trailer*. 
• **GenHead** – the general header contains information such as time of dataset creation and its Iredes version.
• **SiteHead** – the site header contains optional header information for individual machines. For example, machine identifications and working locations.
• **Application objects** can be
  o **Application profile general data** storing parameters from the specified equipment profile.
  o **Optional data** stored optional parameters for the equipment.
• **GenTrailer** – the general trailer is used to secure data integrity with a checksum.

**XsdDbBinder**

The Iredes standard provides templates for the long time storage. XsdDbBinder automatically translates Iredes standard templates into corresponding table structures (schemas) in a relational databas.

### 2.2 XML in Iredes[^2]

XML (eXtensible Markup Language) is a general mark-up language that structures and organise data. XML has a plain textual structure but it is not meant to be easy for humans to read. XML is a W3C recommendation and widespread in a standardised way.
2.3 XML-Schema

A well formatted XML document has one root element where each sub-element has a start and a stop tag. Each Sub-element can be nested from a tree of elements under a root. An XML document is well formatted if some key points are fulfilled:

- It has to use legal Unicode characters.
- It must have a single root element.
- Each element must have a start and a stop tag that have to match exactly.
- The elements must be nested correctly, with no element missing and no overlapping elements.
- The elements are case sensitive.

Validation means to check that an XML-document is consistent according to a pre-defined XML schema. An XML document has to be well formatted, but need not be a valid XML document. However, to allow for standardized data exchanges the XML document should be validated by checking for conformance according to an XML Schema (.XSD - files) describing a standard XML based data exchange format. XML-schemas are also written in XML.

An XML Schema is an abstract group of meta-data containing of a set of schema components; elements, attribute declarations, and complex and simple type definitions.

An XML Schema has some specific guidelines:

- It defines which elements are child elements.
- It defines the structure and relationships of the children and the number of children.
- It is described whether an element is empty or has some content.
- It defines what elements and attributes can be in a valid XML document.
- It defines data types for elements and attributes.
- It sets default and fixed values for elements and attributes.

XML-Schema is selected for the Iredes standard because it can be understood by schooled and non-schooled programmers, and easily be processed in computer applications.

In XsdDbBinder general Iredes XML Schema models are used as templates for generating the corresponding relational database schemas.

2.4 Relational database model

A database is a collection of related data, like data from a company. This data is persistent, i.e. the data does not disappear when the computer is shut down. A database has a description of the data in the database, called a schema. A database managed by a system called a database management system, DBMS, for example Oracle and MySQL.

Data is stored in a database managed by a DBMS because:

- The data is thereby made persistent.
- The DBMS allows complicated searches are done in a simple manner through a query language such as SQL.
- Databases are flexible by allowing easily changes to the database schema.
- The DBMS provides concurrency so that many people can search and update the database at the same time.
• The DBMS provides recovery mechanisms to restore the database if the computer goes down.
• The DBMS provides security routines for computer trespassing.

2.5 Relational database
A relational database \cite{2}, \cite{3} is a collection of records stored in tables. A relational database is described by a schema. The schema describes the tables in the database and the relationships between tables.

**Table DRPPBOOM with a hypothetic key BoomId**

<table>
<thead>
<tr>
<th>Tablename</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRPPBOOM</td>
<td>TIMETAG, BOOMID, LENGTH, TIMEDATED, NUMIPAUTO</td>
</tr>
<tr>
<td>2002-02-01</td>
<td>1, 10, 22</td>
</tr>
<tr>
<td>2004-11-21</td>
<td>81, 10, 22</td>
</tr>
<tr>
<td>2004-11-22</td>
<td>20021, 800, 24</td>
</tr>
</tbody>
</table>

**Figure 2-4 Relation table definitions**

The formal relational database terminology:
• A row in a table is called *tuple*
• A table column header is an *attribute*
• A table is called *relation*
• The data type (type of values) in each column is a *domain*

The values in the tables can’t have more than one value each, so there can’t be two time values in the timetag. Each value in the domain is undividable (*atomic*) and has no further sub-attributes.

**Constraints**

In the relational model there are a number of constraints or restrictions on the values in the database. A schema-based constraint is a type of constraint that can be expressed in the relational database. This includes domain constraint, key constraint, constraint on null, entity integrity constraint, and referential integrity constraint.

*Null* is a value that can be used if you don’t know the current value of if there are no value for some attribute in some row.

A *key* is one or more attribute values that uniquely identify a row in a table. For example, *BoomId* can’t have the same value in the tuples. The value of a key can’t be null.

**Primary key**
To define Primary key you have to know what a minimal super key is.
A super key is one unique attribute or a combination of unique attributes in one table (relation). That means the combination of all attributes is always a super key.
A minimal super key is a candidate key, and a candidate key is a super key, where you can’t take away attributes, still to be unique.

There is always at least one super key and at least one candidate key in each table. Then one of the candidate keys is chosen to be the Primary key.
**Foreign key**
A foreign key is a reference attribute. A reference attribute (foreign) refers to a primary key in another table. The domain (values) in the referencing relation has to be the same as the referenced relation.

<table>
<thead>
<tr>
<th></th>
<th>TIMEHDRUN</th>
<th>BOOMID</th>
<th>LENGTHDRILLED</th>
<th>NUMHFAUTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1900-01-01</td>
<td>1</td>
<td>10</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>1950-11-21</td>
<td>81</td>
<td>10</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>2006-11-29</td>
<td>20033</td>
<td>500</td>
<td>34</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2-5 Foreign key**

In table 2 the primary key is BOOMID.

<table>
<thead>
<tr>
<th>_lonely_column</th>
<th>_lonely_column</th>
<th>_lonely_column</th>
<th>_lonely_column</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BOOMID</td>
<td>TIMEHDRUN</td>
<td>TIMEPOS</td>
</tr>
<tr>
<td>1</td>
<td>1900-01-01</td>
<td>01:01:01</td>
<td>11:00:00</td>
</tr>
<tr>
<td>81</td>
<td>1950-11-21</td>
<td>01:01:01</td>
<td>11:00:00</td>
</tr>
<tr>
<td>20030</td>
<td>2006-11-29</td>
<td>01:01:01</td>
<td>05:00:00</td>
</tr>
<tr>
<td>20033</td>
<td>2006-11-29</td>
<td>02:01:01</td>
<td>06:01:01</td>
</tr>
<tr>
<td>20033</td>
<td>2006-11-29</td>
<td>03:01:01</td>
<td>09:01:01</td>
</tr>
<tr>
<td>20033</td>
<td>2006-11-29</td>
<td>04:01:01</td>
<td>08:01:01</td>
</tr>
</tbody>
</table>

**Figure 2-6 Foreign key**

In BoomTimingData (figure 2-5) the foreign key is BOOMID. The foreign key refers to BOOMID in the table DRPPBOOM (figure 2-6).

To maintain the consistency among tuples between two tables, there is a referential integrity constraint defined. This means that a tuple can’t be removed as there still is data in another table that assumes the tuples existence.

A different constraint can restrict the attribute or domain values by using the CHECK clause. The constraint name is used to identify a particular constraint in case the constraint has to be dropped or be replaced with another constraint. Giving names to constraints is optional.

**Constraint using the CHECK clause**

CONSTRAINT CMM_DRPPBOOM CHECK ( BOOMID >= 1 AND BOOMID <= 100)

**Figure 2-7 Constraint**

This CHECK clause restricts BoomId numbers to be minimum 1 and maximum 100. The constraint name is CMM_DRPPBOOM.

**Normalization**
To get a useful design for the database, use normalization. Normalization is a good approach for not getting the same data stored in many places or maybe the data will not at all be stored.

- The first normalization form 1NF:
  - Each attribute can only have one (atomic) value.
- The second normalization form 2NF:
  - 1NF and each non-key attribute are dependent of the primary key.
- The third normalization form 3NF:
  - 2NF and all non-key attributes must be directly dependent of the primary key or the non-key attribute must not be dependent of other non-key attributes.
**RDB in the system**
Relation database is used as the longtime storage.

**2.6 The choice of translator from XML-Schema to java classes**

For the translation from an XML-Schema to a Java class, one of the demands from IREDES organization was that the translator has to be free-of-charge. There were three tested XML-Schema to class translators XmlBeans\(^4\), Oracles XML developer kit called XDK-package\(^6\), and an open source translator called Xsd2db \(^5\).

There were not all free-translators that could translate the Iredes template, thus there was some testing of different translators found on the web.

**XDK-package (Oracle XML Developer Kit)**

XDK-package is a set of tools components and utilities for C++ and Java objects stored interfaced with Oracle databases. The XDK-package simplifies the usage of XML application. In this developer kit there is an automatic translator for XML-Schema to Java class. The IREDES XML-Schema DrillRig.xsd was tested with the XDK-package.

The problem with the XDK-package is that the XDK-package was not able to translate the Iredes XML-schema. When running the Iredes template in the XDK-package the package gets “serious errors”. This means that the XDK-package translator could not be used for this project.

**Xsd2db**

XSD2DB is an automatic translator from an XML-Schema to Java classes. The IREDES XML-Schema DrillRig.xsd was tested with XSD2DB. Figure 2-10 shows the error code from the XSD2DB program when running the Iredes XML-schema DrillRig.

![Figure 2-8 Error code XSD2DB](image)

This means that the XSD2DB translator could not be used for this project.

**2.7 XmlBeans**

XmlBeans\(^4\) is a technology where XML data is bound to Java interfaces. XmlBeans automatically creates Java classes from XML-Schema (.xsd) files. The main purpose with XmlBeans here is to access valid XML document with Java. Figure 2-8 illustrates how XmlBeans creates classes from XML-Schema. Iredes Profile is the incoming XML-Schema; Class is the classes that XmlBeans create; Process is the XsdbBinder made processes for making SQL-scripts and running of the scripts against the tables in the database. Table is the generated tables in the database.

XmlBeans managed to run the whole Iredes-template and was chosen for implementing XsdDbBinder.
3 The XsdDbBinder system

From an existing Iredes XML-schema the XsdDbBinder system automatically creates a relational database. This system does not populate the database, just creates the schema in the database. Figure 3-1 shows the translation of the content from the Java forms translated into database from

<table>
<thead>
<tr>
<th>Original form</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java classes</td>
<td>Table name</td>
</tr>
<tr>
<td>Java methods</td>
<td>Column name</td>
</tr>
<tr>
<td>Java return types from methods</td>
<td>Column types</td>
</tr>
<tr>
<td>Non Java return type from methods</td>
<td>The name of the child table</td>
</tr>
</tbody>
</table>

Figure 3-1

Figure 3-2 shows the structure of the Iredes template XML-Schema, before translation. Figure 3-3 shows the structure of the corresponding Iredes relation database tables, after translation.

To get a good design for the relational database, there are some rules to follow (see 2.5 Normalization). When the conversion between an XML-tree and a relation database is done the maintaining of the tree structure described in Sect. 3.4 is important. An XML-tree only has data in the leaves. The XML tree structure is preserved by leafs transformed to columns and nodes to tables.

The approach for XsdDbBinder to generate the schema and create the database is illustrated by Figure 3-4. The boxes with the red frames are steps where the program takes care of the translation of the XML-Schema to XmlBean-classes to an SQL-script to create the tables in the database. The black boxes mean the input XML-Schema script and the final generated relational database schema.
The following code is called when creating an Iredes database from an Iredes Template:

```xml
<xsd:element name="DRMaint">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="IR:IRMaintGenType">
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

```java
public interface DRMaint extends org.iredes.xml.IRMaintGenType {...}
public interface IRMaintGenType extends org.iredes.xml.IREDESType{...}
```

Figure 3-5 Example of translation from Iredes template to classes made by XmlBeans

- XmlBeans makes java classes from the standard Iredes templates.

<table>
<thead>
<tr>
<th>TABLE NAME</th>
<th>COLUMN NAME</th>
<th>TYPE NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRMaintDocument$DRMaint</td>
<td>DRMaintDownwCompat</td>
<td>String</td>
</tr>
</tbody>
</table>

Figure 3-6 Example on how the XsdDbBinder extracts the classes into table and column names

```sql
create table DRMaint( IRindex number , DRMaintDownwCompat varchar2(40), DRMaintVersion varchar2(40))
```

Figure 3-7 Example on how the XsdDbBinder makes an ordinary SQL-script

- XsdDbBinder creates the database by generating SQL scripts made via XmlBeans classes. For example, the XML-Schema structure in Fig. 3-4 is translated to the relational database tables in Fig. 3-5 by generating the SQL script in Fig. 3-6. The relational database schema is constructed by a DBMS running the generated SQL script.

Figure 3-7 illustrates how, in the future, the data from an Iredes vehicle is supposed to convert the data stream and long term store the data in the generated relational database.
IMS (Intelligent Management System) is a system for processing data extracted from a vehicle using Iredes.

The DBLoader (Data Base Loader) populates the relational database.

The purpose of XsdDbBinder is to automatically generate the relational database schema, given the XML Schema used for representing data extracted from the vehicle. The import application is a future feature and is currently not part of XsdDbBinder.

1. The data flow starts with signals of data values measured in a vehicle.
2. The data values are digitalized and transformed to the Iredes standard by the IMS. The data is temporarily stored in an XML file validated by an Iredes XML-schema. This guarantees data correctness to some extent. The XML file is stored in the file catalog of the IMS.
3. The DBLoader populates the database with data from the IMS files.

Figure 3-9 shows how XsdDbBinder makes three main steps before the database is ready to use.

Find metadata: The metadata are extracted from the XmlBeans made classes.

Create scripts: The create metadata is transformed into SQL-scripts. These scripts will make tables, columns and constraints between tables for the conserving of the XML tree structure.

Run scripts: The generated SQL scripts are run by the DBMS to create the relational database schema elements.

DB is the final created Iredes relational database ready to be populated.

Figure 3-9: Program steps in XsdDbBinder
3.1 Find metadata

The XmlBeans made classes have to be found in the file system before the metadata to the database can be created. The metadata are extracted from the Iredes profiles though the corresponding XmlBeans classes as illustrated by Fig. 3-10. The table names are extracted from the XmlBeans jar-file by reading the class name from the jar-file.

<table>
<thead>
<tr>
<th>XSD-file</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Table name</td>
</tr>
<tr>
<td>Methods</td>
<td>Column name &amp; Type name</td>
</tr>
</tbody>
</table>

Figure 3-10 Find metadata flow

3.1.1 Path

To get access to the XmlBeans classes and methods, the system has to know where in the computer file system the classes are placed. From the Iredes template (Xml-Schema files) the targetPath (where in the computer file system the classes are placed) is computed by extracting the namespace URI from the Xml-Schema. When there is no name in targetPath, XmlBeans sets the catalogue to “noNamespace”. Therefore XsdDbBinderhardcodes noNamespace as the search path when the targetPath in the template is empty.

3.1.1.1 Table name

The table names for the relational database are extracted from the class names found in the jar file created by XmlBeans. The extraction is made by using the package java.util.jar.

3.1.2 Columns

The column names for the Iredes database are extracted from the XmlBeans generated method names in the jar file, and the type for the columns are taken from the return types from these methods. Figure 3-11 shows what methods come out from XmlBeans.
java.lang.String *getEvent();*
org.iredes.xml.IRtextLong *xgetEvent();*
void *setEvent*(java.lang.String event);
void *xsetEvent*(org.iredes.xml.IRtextLong event);

Figure 3-11, method name, for making column

From one node in the xml-schema, XmlBeans generates some extra methods that are not required for making columns from methods. In figure 3-10 there is a method called *getEvent(),* that is coloured yellow, the xml-node is called Event. In the figure there are some extra methods with the name Event that are unnecessary for this assignment. This assignment only needs the name of the node (method).

One redundant method starts with “set”, another method starts with “get” and both of these two methods also have a duplicate method that starts with an “x”.

The get-method is chosen because the return type from the methods is also important. There has to be some connection between the Xml-Schema type, java type and Oracle type for making the new database. Of the return types from the XmlBeans made classes it is only the get-method that has the corresponding return type used in the generation of the relational database schema. The rest of the information is filtered by the XsdDbBinder method *getClassMethodNames.*

### 3.1.2.1 Get SQL types for SQL script

The SQL types correspond to the return type from the return Java types of the XmlBeans generated get-methods. The java types are translated into Java by the SQL mapper, IRJava2OracleMapper. Figure 3-13 is the translation table for XsdDbBinder.

<table>
<thead>
<tr>
<th>Java Type</th>
<th>SQL Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Varchar</td>
</tr>
<tr>
<td>BigDecimal</td>
<td>Number</td>
</tr>
<tr>
<td>Boolean</td>
<td>Number</td>
</tr>
<tr>
<td>Byte</td>
<td>Number</td>
</tr>
<tr>
<td>Short</td>
<td>Number</td>
</tr>
<tr>
<td>Type</td>
<td>Java Type</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>Int</td>
<td>Number</td>
</tr>
<tr>
<td>Long</td>
<td>Number</td>
</tr>
<tr>
<td>Float</td>
<td>Number</td>
</tr>
<tr>
<td>Double</td>
<td>Number</td>
</tr>
<tr>
<td>byte[]</td>
<td>Raw</td>
</tr>
<tr>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>Time</td>
<td>Date</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Timestamp</td>
</tr>
<tr>
<td>Calendar</td>
<td>Date</td>
</tr>
</tbody>
</table>

Figure 3-12 Conversion table Java vs. SQL types

### 3.1.2.2 Get constraints between tables

Figure 3-12 shows how different Java return types from the get-methods are mapped to corresponding SQL datatypes. For return types not in table in figure 3-12 the return type is assumed to a reference to a child table.

Figure 3-14 shows how the return type from the get-methods look like when it is not a standard java type in Figure 3-12.
public interface DRMaint extends org.iredes.xml.IRMaintGenType {
    org.iredes.xml.drillRig.DRMaintDocument.DRMaint.EventLog getEventLogArray();
    ...
}

Figure 3-14 Non-Java standard return type in java code

In Figure 3-14 The XmlBeans generated get-methods that are marked in yellow or red are methods that XsdDbBinder uses for making the database tables. The one not marked are ignored.

Figure 3-15 shows the XsdDbBinders corresponding tables from the classes shown in figure 3-14. Each line has information of one column, table name and type name for the column. The string for table name contains a path where in the tree system the column is placed. In the figure the first column Event has type string, and the column is placed in the table EventLog. DRMaint is EventLog’s parent. DRMaintDocument is the documentation for the tree, and is not the root.

<table>
<thead>
<tr>
<th>TABLE NAME</th>
<th>COLUMN NAME</th>
<th>TYPE NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRMaintDocument$DRMaint$</td>
<td>EventLog</td>
<td>String</td>
</tr>
<tr>
<td>DRMaintDocument$DRMaint$</td>
<td>EventLog</td>
<td>Calendar</td>
</tr>
</tbody>
</table>

3-16
### 3.1.3 Creating and running the SQL Scripts

Figure 3-16 shows the table for EventLog, the same table as in figure 3-15.

```sql
create table EventLog( IRindex number , Event varchar2(40), TimeTag date)
```

The four next steps show the order of the SQL scripts for how the tables are done. For each table there must be created a table and then constraints.

```sql
create table DRMaint( IRindex number , DRMaintDownwCompat varchar2(40), DRMaintVersion varchar2(40))
```

1. The script for making the table is created.

```sql
alter table DRMaint add ( Iref number )
```

2. A column for primary key is added.

```sql
alter table DRMaint add primary key ( Iref )
```

3. The primary key script is set.

```sql
alter table GenTrailerType add constraint fk_GenTrailerType foreign key ( IRindex ) references DRMaint( Iref )
```

4. The foreign key script is set.

### 3.1.4 Executing the SQL Script

This section will show how XsdDbBinder runs the SQL scripts, and how the database will be normalised after the transformation from XML schema to a database ready for population.

To maintain a tree structure there are constraints between tables, and to keep a useful design for the database XsdDbBinder method uses normalization. To get the tables normalized all normalization forms have to be fulfilled. 1 NF is fulfilled since the xml data is atomic. Figure 3-17 shows the constraint SQL script for the foreign key. The relationship between tables is a reference constraint, foreign key, and in the figure the constraint name is FK_IROPTIONTYPE.
**CONTRAINT "FK_IROPTIONTYPE" FOREIGN KEY ("IRINDEX") REFERENCES "DRMAINT" ("IREF")**

Figure 3-17 Constraint script

Figure 3-18 shows the index columns. For finding the right data in the tables there is a column for index needed. XsdDbBinder makes one index column for each table, in figure the index columns are marked with read and is called IRINDEX.

![Index Columns](image)

**Figure 3-18 The index column**

Figure 3-19 shows how a primary and foreign key are created by XsdDbBinder. To get the primary key the column IREF, marked in red in the table DRMAINT, is created, IREF have unique values. The reference for primary key, marked in red in the table DRMAINT. In figure 3-18 the reference FK_IROPTIONTYPE, marked in purple, in the table IROPTIONTYPE is a foreign key. The foreign key in table IROPTIONTYPE is the column IRINDEX and this foreign key refer to the primary key, the reference is marked in yellow, IREF, the column is marked in red, in the table DRMAINT.

![Primary Key and Foreign Key](image)

**Figure 3-19 Primary key and foreign key**

2 NF is fulfilled because each attribute has one value and all non-key attributes are dependent on the primary key. The primary key is created by a new unique column (IREF) for each table (DRMaint and IROPTIONTYPE).

3NF is fulfilled by making the new IREF column; this means none of the non-key columns is dependent of any other non-key. In table DRMaint in figure 3-19 the columns DRMAINTDOWNWCOMPAT and DRMAINTVERSION are not dependent on each other and are non-keys but they are dependent on IREF that is the primary key.

There are some ordering issues with the creation of the SQL scripts. These problems are solved by sorting definitions in the scripts with the following rules.
1. Create a table, since the table has to be created first.
2. Add a reference attribute to the table, since the reference attribute must be created after the table because a table has to exist when an attribute is added.
3. Create a primary key in table, since the primary key must be set after the attribute and before the foreign key, and because a foreign key must have a primary key before it is created.
4. Finally create foreign key constraints.

### 3.2 Normalization in XsdDbBinder

For the tree structure from XML-schema this will be the case:

- 1NF is fulfilled because all leaves are atomic by definition in the XML-schema.
- For XML the 2NF and 3NF is fulfilled because:
  
  **For Iredes XML-schema**
  
  - 1NF is fulfilled because all leaves are atomic by definition in the Iredes XML-schema.
  - 2NF is fulfilled because 1NF and all non-key attributes are dependent of the primary key.
    By creating a new column for the foreign key and primary key the 2NF will be automatic guaranteed without any interference of the Iredes template.
  - 3NF is fulfilled because 2NF and none of the non-keys depend on any other non-key.

### 3.2.1 Schematized flow of how normalization are handled in the XsdDbBinder process

Figures 3-20, 3-21 and 3-22 shows the schematized flow of the different states the schema is in. 3-20 shows the XML-schema format, 3-21 shows the in-between class format and 3-22 shows the database format.

Figure 3-20 shows the Iredes Xml-schema tree GenHead (without relations). The root is DRMaint to the left and the leafs are at the end of the branches to the right.
Figure 3-20 Xml-schema

Figure 3-21 show the XmlBeans generated classes. Each square is a class; the class name is in the blue box in each square. The name of each method and the return type is in the white field in each square; the return type to the left and method name to the right. The root DRMaint, in figure 3-20 is made a class in figure 3-21. In figure 3-21 the class EventLog is translated from the branch EventLog in figure 3-20. The methods getTimeTag() and getEvent() in the class
EventLog in figure 3.21 are translated from the XML-schemas branch EventLogs leafs TimeTag and Event, in figure 3.20.

Figure 3.21: The XmlBeans made classes.
Figure 3-22 shows the XsdDbBinder made relation tables. DRMAINT is translated to a table in figure 3-22, EVENTLOG is translated into a table and EVENT and TIMETAG are translated into columns in the EVENT table.

![Figure 3-22 The created database tables](image)

### 3.3 Method for handling Any-Option

An any-option is a special type for IREDES. An any-option is a type where you can add any optional vendor specific or additional XML schema on to the XML schema. This type is translated to a table and added an extra column as a primary key, for future tables and future foreign keys.

Figure 3-23 shows how the any-option type looks like in the IREDES XML schema. The any-option is a type that is defined in the IREDES standard as a complex type. The upper pane in
figure 3-23 shows the IRoptionType definition in the IRTypes-schema. The lower pane shows the call to the type from the DrillRig-schema.

```xml
<xsd:complexType name="IRoptionType">
  <xsd:element name="OptionSchema" type="xsd:anyURI" minOccurs="0"/>
</xsd:element>
</xsd:complexType>

<xsd:element name="Options" type="IR:IRoptionType" minOccurs="0"/>
</xsd:element>
```

Figure 3-23 Any-option in Xml-schema

In figure 3-24 the IRoptionType has been translated to a class name and the XML element OptionSchema has been translated to the method OptionSchema.

Figure 3-24 XmlBeans made class

Figure 3-25 shows how XsdDbBinder handles IREDES any-option type. In figure 3-25 the column IREF is added to the table IROPTIONTYPE and the column IREF is made primary key.

Figure 3-25 the translated Option in relation table format

Figure 3-26 shows a hypothetic case of a new added table called MIDDLE. The reference foreign key, marked with purple in the table MIDDLE, refer to the primary key in IROPTIONTYPE.
Figure 3-26 the XsdDbBinder translated Any-option in relation table format, with a hypothetic table called MIDDLE

Figure 3-26, shows the script for the translation

```
Alter table MIDDLE add constraint FK_MIDDLE foreign key (IRINDEX)
    references IROPTIONTYPE (IREF)
```

Figure 3-27 Script for vendor reference

4 Conclusions

XmlBeans is the only tested system that could convert the complex Iredes profiles into java classes. One thing missing in XmlBeans is the conversion of the xml type hexBinary to java standard type.

There is no need to manually change the original Iredes templates for generating a corresponding relational database, since

- XsdDbBinder automatically generates a normalized relational database.
- XsdDbBinder takes care of the IREDES Any-Option.
- XsdDbBinder traps the XmlBeans error from the missing XML type hexBinary.

You do not have to have any knowledge of XML to add new features in XsdDbBinder, thanks to XmlBeans.
5 Tutorial program

Install the program.
Unzip XDBind.rar in the catalogue where you want the program.
Figure 9-1 shows how the unpacked files look like.
- *Iredes* catalogue contains the Iredes template, and the XmlBeans made jar file specially made from Iredes template (DR.jar).
- *Log* contains the log-files. These log-files are used when executing the program.
- *Oracle* contains the catalogue for oracle drivers (included in this package).
- *Org, repackage* and *schema_apache_xmlbeans* are three XmlBeans specific catalogues; NB! Do not remove these three catalogues. The catalogues contain information that is used when running XmlBeans commands in the Xdbind program.
- *Xdbind.exe* is the program.

![Figure 5-1 program catalogue](image)

Figure 9-2 shows the menu when typing xdbind, which is the execute program command, or the help command.
The program is executed in windows command prompt.

![Figure 5-2 screenshot](image)

**XDBIND COMMAND**

xdbind command runs the program XsdDbBinder
Xdbind <options>

Options
<table>
<thead>
<tr>
<th>Option</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>-xsd</td>
<td>The path where the XML-Schema is located. The whole path is needed.</td>
</tr>
<tr>
<td>-jar</td>
<td>The path to the XmlBeans made jar files.</td>
</tr>
<tr>
<td>-driver</td>
<td>Name of the driver. Default: oracle.jdbc.driver.OracleDriver</td>
</tr>
<tr>
<td>-host</td>
<td>URL, hostname, location of the database.</td>
</tr>
<tr>
<td>-id</td>
<td>Database name. Default: localhost</td>
</tr>
<tr>
<td>-user</td>
<td>User name. Default: XE</td>
</tr>
<tr>
<td>-password</td>
<td>Password.</td>
</tr>
<tr>
<td>-h or --help</td>
<td>Help command.</td>
</tr>
</tbody>
</table>

Figure 9-3 shows example of the command path for running the program. When the text “Running…” appears the program is creating the database. When the text “The IR-Binder is done.” appears the database is ready. Press any key and you can populate the database.

![Figure 5-3 a successful execution](image)
6 References


