TU Kaiserslautern
FB Informatik, AG HIS
Prof. Dr.-Ing. Stefan Deßloch
M.Sc. Michael Hohenstein

Worksheet 3 - DB Schema Design and Programming

"JDBC, Schema metadata, XML Schema"

Introduction

Goal. In this workheet, you will learn how to retrieve metadata about a database schema, as well as the basics of XML Schema concepts like data type definition, complex types and integrity constraints.

During the last decade, XML has become a central standard in data-centric (web) applications. Therefore, it becomes more and more important to make existing operational data from relational databases also available to new applications, such as Web services, that rely on semi-structured data. Such an *XML view* on relational data requires the definition of an appropriate mapping between these two data models. A data mapping defines how data that is available in a so-called *source schema* is represented in the *target schema.*

In this worksheet you will develop a tool that generates the target schema by mapping the relational source schema to a semi-structured (XML) target schema defined in *XML Schema.* The mapping itself is defined by a set of given mapping rules.

Converting relational data into XML

In exercise 2, you must generate an XML Schema definition from the relational schema fetched in exercise 1. Before that, we must understand how to map the relational *data* itself into XML data. The rules for converting a relational database into an XML document are given below.

Note: You **must not** implement a tool to export relational data into XML. The goal of these rules is just to explain how the exported data **would look like**.

- The root element of the resulting document is named after the schema. It contains the exported rows of all tables as children. This means that there is no explicit XML node that directly represents a table.
- Each row is exported as an element named after the table it belongs to. We refer to it as *row element.*
- A column can be represented in three ways, depending on its characteristics:
  - Columns that serve as primary keys of the source table are exported as an attribute of the row element they identify. For this exercise, we ignore primary key definitions on multiple columns, so there is at most one such attribute for every row element.
  - If a column has a foreign key constraint (again, we ignore compound keys), then its column element has an attribute with the name `ref` that holds the value of the foreign key.
  - Columns that do not participate in any primary or foreign key definition are exported as child elements of the row they belong to. We call these elements *column elements.* The actual data is then stored as a text node under the column element.
- Columns that contain NULL are simply omitted from the XML representation, i.e., there is no explicit representation of NULL values.

- If exactly one of the columns of a table B has the NOT NULL constraint and it is a foreign key that references a table A, then we say that B is *existence-dependent* on A. Row elements of an existence-dependent table are not placed directly under the root element, but instead as a sub-tree of the row element which they refer to through the foreign key relationship.
- Because the relationship between an existence-dependent table and its parent is already encoded as a parent-child relationship in the XML tree, there is no need to explicitly represent the foreign key column and its value. Therefore, it is omitted from the row element.

To clarify how the rules work and what kind of documents they produce, we refer to the examples in the Wiki.

Exercises

Exercise 1: Retrieving metadata from a database

Develop a Java program that uses JDBC to collect information about all tables, their columns (name, data type, and precision or length), and defined integrity constraints in a given database schema.

Similar to exercise 3 of worksheet 1, we provide a basic JAR package with bean classes and a `Main` class which runs the program. Your task is to implement the method `getSchemaInfo()` in the interface `SchemaRetriever`. The retrieved schema information must be delivered as an instance of the `Schema` bean class, which contains information about each table in the schema. The `Main` program provided will display the information contained in this object on the screen using the provided `toString()` method. Therefore, you again do not have to implement serialization to strings.

For simplicity, you only have to consider *primary key*, *unique*, *not null*, and *referential constraints*. You must not consider any insert, update, or delete rules of referential constraints or any other constraint types such as table check constraints. Table 1 shows how the output would look like for the `student` table of the sample schema (see Wiki). Note that your implementation must generate this output for *all* tables of a schema, and not for a single table.

```
TABLE student (
  studentnumber INTEGER(10) UNIQUE NOT NULL PRIMARY KEY
  firstname VARCHAR(255) NOT NULL
  lastname VARCHAR(255) NOT NULL
  supervisor BIGINT(19) REFERENCES academic(id)
  assistant CHAR(7) REFERENCES course(code)
)
```

Table 1: Sample Output

Relational systems provide information about defined schemas, tables, and columns in the views/tables of the so-called *system catalog*. In addition to the system catalog, JDBC also provides metadata about the database through the `DatabaseMetaData` object.

Exercise 2: Generating an XML Schema definition

Extend your program so that it generates an XML Schema definition to validate the XML data exported from a relational database. A nice property of XML Schema is that it can be defined itself using XML, in what we call an *XSD document*. The solution must be implemented in the `getXMLSchema()` method of the `SchemaConverter` interface. The method takes the `Schema` object of exercise 1 and returns an `org.w3c.dom.Document` instance for the XSD document.

Below, we provide a brief specification of how the mapping rules given previously can be represented in XML Schema. Please refer to the documentation on XML Schema for understanding each of its constructs.

Again, we emphasize that you **must not** implement a tool to export relational data into XML. Your program just has to **validate** XML data that was **already exported**.

- A *complex type* is used to specify row elements as well as the root element of the document. These complex types must be *global*, i.e., occur directly below the root element in the XSD document and are referenced by a name. The name is formed by taking the table name and appending the word “Type", e.g., `studentType`.
- The column elements of a row are specified as *local type definitions* inside the complex type of the row. A local type definition is simply an unnamed complex type whose instances can only occur under an instance of the parent type.
- Column elements must occur in the same order as defined by the `Schema` object. To ensure the ordering, you must define the column complex types inside a `xs:sequence` construct.
- The data type of a column element is specified by a global *simple type*, whose name is composed of the table name, the column name, and the word “Type", e.g., `studentFirstnameType`. All composed type names must follow the *camel case* convention (i.e. *longComposedWord*).
- Simple types defined for columns are restrictions of their equivalent base types in XSD. Use the following mapping between SQL and XSD types:

BIGINT → `xs:long`	DATE → `xs:date`
INT → `xs:int`	TIMESTAMP → `xs:dateTime`
SMALLINT → `xs:short`	CHAR → `xs:string`
REAL → `xs:float`	VARCHAR → `xs:string`
DOUBLE → `xs:double`	

  Note that for CHAR and VARCHAR, the specified length (e.g., VARCHAR`(28)` must also be reflected in the XSD simple type.
- Primary key and NOT NULL constraints must be specified as occurrence indicators in the element specification of a column. Referential constraints must be specified as `xs:key` and `xs:keyref` specifiers within the complex type of the root element.

More important than accepting valid document instances, such as those produced by a correct implementation of exercise 2, your schema definition must be able to *reject* invalid data, i.e., data that violates the constraints defined in the original relational schema. The provided `Main` class can also be used to validate documents and test your solution locally before submission.

Once again, we refer to examples in the Wiki to clarify the mapping process.

Submission

Submission deadline is Sunday, June 30, 2019 at 23:59:59.
Lots of fun and success!