

HShreX basic shreddings

This document describes the basic shredding rules and the annotations that can be used to influence said shredding rules. This is version 0.4.1 of this document, dated June 3rd 2009.

Elements

Complex elements are shredded to tables, and complex elements that can have text data (simple content) also gets a special column called "nodeValue" for storing that data. Simple elements that can occur at most one time under their parent become fields (columns) in their parent table. Attributes also become fields in their parent table. If a simple element or attribute is optional, their corresponding column becomes nullable. By default, table names are constructed using the path to the element (similar to XPath but an underscore is used as a separator instead of a forward slash). Field names are constructed using the name of their corresponding element or attribute. Here's a simple example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="movies">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="movie" type="movieType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="movieType">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="actor" type="actorType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="actorType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="previousProduction" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="age" type="xs:positiveInteger" use="required"/>
  </xs:complexType>
</xs:schema>
```

The root element "movies" is a complex element that has no simple child elements or attributes and since it's the root element it doesn't have a parent element. It will get mapped to table named "movie" and it will only get a single, automatically generated, column named shrex_id to act as primary key. All tables get this column. The complex element "movie" contained by the root element will be mapped to a table named "movies_movie". It has two children: "title" and "actor". "title" is a simple element and that must occur once and only once so it will be mapped to a non-

Nullable field in the "movies_movie"-table. The second child, "actor", is a complex element so we will get a third table, this one named "movies_movie_actor". Under "actor" we find two simple child elements and an attribute. All these must be used and they can only be used one time under a given "actor"-element so we will get three non-Nullable fields in the table

"movies_movie_actor": "name", "previous_production", and "age". Here's the complete CREATE TABLE-script:

```
CREATE TABLE movies (  
    shrex_id INT NOT NULL,  
    PRIMARY KEY(shrex_id)  
);
```

```
CREATE TABLE movies_movie (  
    shrex_id INT NOT NULL,  
    shrex_pid INT NOT NULL,  
    title VARCHAR(250) NOT NULL,  
    PRIMARY KEY(shrex_id),  
    FOREIGN KEY(shrex_pid) REFERENCES movies(shrex_id)  
);
```

```
CREATE TABLE movies_movie_actor (  
    shrex_id INT NOT NULL,  
    shrex_pid INT NOT NULL,  
    age INT NOT NULL,  
    name VARCHAR(250) NOT NULL,  
    previousProduction VARCHAR(250) NOT NULL,  
    PRIMARY KEY(shrex_id),  
    FOREIGN KEY(shrex_pid) REFERENCES movies_movie(shrex_id)  
);
```

But say we let the simple element "previousProduction" under "actor" occur zero to an unbounded number of times, instead of requiring that it always must occur one time (no more, no less):

```
<xs:complexType name="actorType">  
    <xs:sequence>  
        <xs:element name="name" type="xs:string"/>  
        <xs:element name="previousProduction" type="xs:string"  
            minOccurs="0" maxOccurs="unbounded"/>  
    </xs:sequence>  
    <xs:attribute name="age" type="xs:positiveInteger" use="required"/>  
</xs:complexType>
```

What effect would it have on the relational mapping? Well, it could no longer be a field in the "movies_movie_actor"-table, since it's now a list. Instead a fourth table would be generated to store "previousProduction", and it would have the "movies_movie_actor"-table as its parent table. Here follows the modified "movies_movie_actor"-table and our new table:

```
CREATE TABLE movies_movie_actor (  
    shrex_id INT NOT NULL,  
    shrex_pid INT NOT NULL,
```

```

    age INT NOT NULL,
    name VARCHAR(250) NOT NULL,
    PRIMARY KEY(shrex_id),
    FOREIGN KEY(shrex_pid) REFERENCES movies_movie(shrex_id)
);

CREATE TABLE movies_movie_actor_previousProductions (
    shrex_id INT NOT NULL,
    shrex_pid INT NOT NULL,
    previousProductions VARCHAR(250),
    PRIMARY KEY(shrex_id),
    FOREIGN KEY(shrex_pid) REFERENCES movies_movie_actor(shrex_id)
);

```

Let's return to our original example where "previousProduction" could occur once and only once under a given parent, but this time we rewrite the complex type "actorType" so it references two groups: one containing its simple elements and an attribute group that contains its single attribute:

```

<xs:complexType name="actorType">
  <xs:sequence>
    <xs:group ref="actorGroup"/>
  </xs:sequence>
  <xs:attributeGroup ref="actorAttributeGroup"/>
</xs:complexType>

<xs:group name="actorGroup">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="previousProductions" type="xs:string"/>
  </xs:sequence>
</xs:group>

<xs:attributeGroup name="actorAttributeGroup">
  <xs:attribute name="age" type="xs:positiveInteger" use="required"/>
</xs:attributeGroup>

```

How would this effect our relational mapping of the complex type "actorType", i.e., what would the table "movies_movie_actor" look like with the schema above? The answer is that it would look exactly the same as in the very first mapping I showed. The group is not visible and it shouldn't be because it's just used to group related elements and attributes, which makes writing readable schemas a bit easier. However, you can annotate a group when it's referenced and that will indeed affect mapping for anything contained in that particular group.

Any

Unfortunately, there's no support for <xs:any>, but there is, however, a workaround: instead of <xs:any>, use a typeless complex element. A typeless complex element will get mapped to XML and HShreX will insert the entire subtree in raw XML form when parsing data files, and that subtree can contain elements not part of the schema itself.

Annotations for Shrex

Tablename

Value: string

Applies to: Any attribute, element or group that is going to be mapped to a table.

Actions: The string is used as the table name instead of the generated one which is based on the path of the entity. Additionally, it can be used to merge tables because if you use tablename to specify the name of an already existing table, those two tables are merged. Merging means that any element that maps to a field that has the same name as an existing will be mapped to the existing field. This is to avoid invalid tables where you have multiple fields with the same name. The characteristics of the field is determined by the original creator, so only merge fields of the same type. All value fields become nullable when two tables are merged. When appropriate, these merged tables have more than one parent table and then they get additional foreign keys. When this happens, all foreign key and value fields become nullable as well as the value fields.

Comments: According to an older document, tablename can be used to force entities to a table, but that is not case. It only works on entities that are going to be mapped to a table anyway. Use shrex:outline in conjunction with tablename to force mapping to a table.

Example 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:shrex="http://www.cse.ogi.edu/shrex">
  <xs:element name="families">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="family" type="familyType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="familyType">
    <xs:sequence>
      <xs:element name="parent" type="parentType" shrex:tablename="familymember"/>
      <xs:element name="child" type="childType" shrex:tablename="familymember"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="parentType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="job" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="childType">
```

```

<xs:sequence>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="school" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:schema>

```

The schema above will merge the two tables families_family_parent and families_family_child under the same table named familymember. Since the two merged tables have the same parent table, the merged table familymember will only retain one set of foreign keys. Here is the resulting relational schema:

```

CREATE TABLE families (
  shrex_id INT NOT NULL,
  PRIMARY KEY(shrex_id)
);

```

```

CREATE TABLE families_family (
  shrex_id INT NOT NULL,
  shrex_pid INT NOT NULL,
  PRIMARY KEY(shrex_id),
  FOREIGN KEY(shrex_pid) REFERENCES families(shrex_id)
);

```

```

CREATE TABLE familymember (
  shrex_id INT NOT NULL,
  shrex_pid INT NOT NULL,
  name VARCHAR(100),
  job VARCHAR(100),
  school VARCHAR(100),
  PRIMARY KEY(shrex_id),
  FOREIGN KEY(shrex_pid) REFERENCES families_family(shrex_id)
);

```

Example2:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:shrex="http://www.cse.ogi.edu/shrex">
  <xs:element name="productions">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="movieProduction" type="movieProductionType"
          maxOccurs="unbounded"/>
        <xs:element name="sitcomProduction" type="sitcomProductionType"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

```

```

<xs:complexType name="movieProductionType">
  <xs:sequence>
    <xs:element name="movie" type="xs:string"/>
    <xs:element name="actor" type="xs:string" maxOccurs="unbounded"
      shrex:tablename="people"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="sitcomProductionType">
  <xs:sequence>
    <xs:element name="sitcom" type="xs:string"/>
    <xs:element name="producer" type="xs:string" maxOccurs="unbounded"
      shrex:tablename="people"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>

```

Here we merge the two tables productions_movieProduction_movie_actor and productions_sitcomProduction_producer under the same table people. Since they have different parents, the table people will contain two foreign keys.

Fieldname

Value: String

Applies to: Attribute or simple element

Actions: The string is used as the column name. Usually used for simple elements or attributes that maps to fields but can also be used with complex elements that are mapped to XML.

Comments:

Examples:

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:shrex="http://www.cse.ogi.edu/shrex">
  <xs:element name="employees">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="employee" type="employeeType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="employeeType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="position" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="workedLong" type="xs:string" shrex:fieldname="worked__long"/>
  </xs:complexType>
</xs:schema>

```

The schema above will generate a table called employees_employee and the attribute workedLong will be mapped to a field in that table and since we're using the fieldname annotation it will be called "worked__long".

```
CREATE TABLE employees (
  shrex_id INT NOT NULL,
  PRIMARY KEY(shrex_id)
);
```

```
CREATE TABLE employees_employee (
  shrex_id INT NOT NULL,
  shrex_pid INT NOT NULL,
  worked__long VARCHAR(100),
  name VARCHAR(100) NOT NULL,
  position VARCHAR(100) NOT NULL,
  PRIMARY KEY(shrex_id),
  FOREIGN KEY(shrex_pid) REFERENCES employees(shrex_id)
);
```

Sqltype

Value: Currently only supports "CLOB", VARCHAR", or "NUMBER". A warning message is displayed for other strings and the annotation is ignored.

Applies to: Attribute or simple element or XML-mapped complex element.

Actions: It overrides the default XML Schema type translation. The internal types set by these are: CLOB: SQL_CLOB, VARCHAR: SQL_STRING and NUMBER: SQL_INT. How these are translated depends on the database connection used, but the simulated connection (FileConnection, that uses DB2-settings) translate these into CLOB, VARCHAR and INT, respectively. Note that this annotation only changes the type used in the generated scripts, it doesn't change how data is read.

Comments: Maybe add more types in the future.

Examples:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:shrex="http://www.cse.ogi.edu/shrex">
  <xs:element name="cars">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="car" type="carType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="carType">
    <xs:sequence>
      <xs:element name="modelName" type="xs:string"/>
      <xs:element name="maker" type="xs:string" shrex:sqltype="NUMBER"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Here we can see that column maker in the table cars_car has type INT, which we forced using the annotation shrex:sqltype=NUMBER.

```
CREATE TABLE cars (  
  shrex_id INT NOT NULL,  
  PRIMARY KEY(shrex_id)  
);  
CREATE TABLE cars_car (  
  shrex_id INT NOT NULL,  
  shrex_pid INT NOT NULL,  
  modelName VARCHAR(100) NOT NULL,  
  maker INT NOT NULL,  
  PRIMARY KEY(shrex_id),  
  FOREIGN KEY(shrex_pid) REFERENCES cars(shrex_id)  
);
```

Fieldsize

Value: Integer
Applies to: Attribute or simple element
Actions: Sets the field size of the attribute. Overrides the default field size.
Examples:
Comments: Currently not implemented.

Identityscheme

Value: KFO, Interval, Dewey, false
Applies to: Root element
Actions: This annotation specifies how keys should be generated. The default key generation scheme (which you get if you do not use identityscheme at all or set it to false) generates numeric keys by maintaining a counter, starting from 1, for each table and simply increments that as new tuples are generated. However, sometimes it's desirable to maintain the tree order in the relational model, and that's where this annotation comes in handy. I will now describe the effect of the specific values this annotation supports.
KFO: KFO generates numeric keys just as the default key generation method does, but instead of having one counter per table, KFO maintains a global counter that is incremented each a time a new tuple for any table is generated. This means no two tuples will ever have the same key. The counter starts at the value 1.
Dewey: Keys in Dewey are generated by counting children under their respective parent and putting together all counters from the root to the given depth to a single key. Each parent/child level is separated by a dot. This means that a key value of "1" indicates the root element (that hasn't got a parent). A key of "1.2" means the second child of the root element and another, more complex example, is "1.3.2.1", which means, from left to right: the first child (1) of the second child (2) of the third child (3) of the root (1). These keys are handled as strings by the database and beware that they can get quite long if you're dealing with very deep structures.
Interval: Interval is another way of generating key that is, supposedly, supported by the old ShreX from 2005. It's currently not implemented but if one uses it, the

validator won't reject it, but one will be informed that the default key generation method will be used instead.

Comments:

Examples: I will now show a schema with a corresponding XML data file and show how the schema is mapped and after that show how the insert script for the given data would look if one used default key generation, KFO, and Dewey, respectively.

XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="movies">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="movie" type="movieType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="movieType">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="year" type="xs:gYear"/>
    </xs:sequence>
    <xs:attribute name="goodMovie" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:schema>
```

XML data file:

```
<?xml version="1.0" encoding="UTF-8"?>
<movies xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
  xs:noNamespaceSchemaLocation="./optional_string_attribute.xsd">
  <movie goodMovie="very good">
    <title>Bladerunner</title>
    <year>1982</year>
  </movie>
  <movie>
    <title>A room with a view</title>
    <year>1985</year>
  </movie>
</movies>
```

Tables generated:

```
CREATE TABLE movies (
  shrex_id INT NOT NULL,
  PRIMARY KEY(shrex_id)
);

CREATE TABLE movies_movie (
  shrex_id INT NOT NULL,
  shrex_pid INT NOT NULL,
  goodMovie VARCHAR(250),
```

```
title VARCHAR(250) NOT NULL,  
year VARCHAR(250) NOT NULL,  
PRIMARY KEY(shrex_id),  
FOREIGN KEY(shrex_pid) REFERENCES movies(shrex_id)  
);
```

Now let's look at the keys generated in the insert script for default key generation, KFO, and Dewey.

Default key generation:

```
INSERT INTO movies (shrex_id) VALUES  
    (1);  
INSERT INTO movies_movie (shrex_id, shrex_pid, goodMovie, title, year) VALUES  
    (1, 1, 'very good', 'Bladerunner', '1982'),  
    (2, 1, null, 'A room with a view', '1985');
```

KFO:

```
INSERT INTO movies (shrex_id) VALUES  
    (1);  
INSERT INTO movies_movie (shrex_id, shrex_pid, goodMovie, title, year) VALUES  
    (2, 1, 'very good', 'Bladerunner', '1982'),  
    (3, 1, null, 'A room with a view', '1985');
```

Dewey:

```
INSERT INTO movies (shrex_id) VALUES  
    ('1');  
INSERT INTO movies_movie (shrex_id, shrex_pid, goodMovie, title, year) VALUES  
    ('1.1', 1, 'very good', 'Bladerunner', '1982'),  
    ('1.2', 1, null, 'A room with a view', '1985');
```

Edgemapping

Value: true, false
Applies to: Element
Actions: If the value is true its descendants are shredded according to edge mapping
Examples:
Comments: Does not work in our version. Prioritized?

Maptoxml:

Value: true, astable
Applies to: Attribute or element
Actions: If the value is true ShreX looks how many times that element can occur under its parent and uses that information to decide if the element needs to be outlined or not (it needs to be outlined if it can occur more than one time under its parent). If the value is astable the element is always outlined, regardless if it needs to be or not. When maptoxml is applied to a complex element, all children of that complex element gets the same mapping as their parent. Also, if a complex type has no type set in the XML Schema, the default mapping is to an XML column (that is outlined if

need be).

Comments: Using tablename to force to a table does not work.

Example 1:

```
<xs:complexType name="employeeType">
  <xs:sequence>
    <xs:element name="id" type="xs:positiveInteger"/>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="address" type="addressType" shrex:maptoxml="true"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="addressType">
  <xs:sequence>
    <xs:element name="street" type="xs:string"/>
    <xs:element name="city" type="xs:string"/>
    <xs:element name="telephone" type="telephoneType" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="validAddress" type="xs:string"/>
</xs:complexType>
<xs:complexType name="telephoneType">
  <xs:sequence>
    <xs:element name="home" type="xs:string"/>
    <xs:element name="work" type="xs:string"/>
    <xs:element name="cellphone" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Here the complex type address is using the annotation maptoxml=true. Since it can only occur once under its parent employeeType, it gets mapped to an XML column in its parent table. If employeeType is mapped to a table staff_employee, we get the following relational schema:

```
CREATE TABLE staff_employee (
  shrex_id INT NOT NULL,
  shrex_pid INT NOT NULL,
  id INT NOT NULL,
  name VARCHAR(100) NOT NULL,
  address XML NOT NULL,
  PRIMARY KEY(shrex_id),
  FOREIGN KEY(shrex_pid) REFERENCES staff(shrex_id)
);
```

Example 2:

```
<xs:complexType name="employeeType">
  <xs:sequence>
    <xs:element name="id" type="xs:positiveInteger"/>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="address" type="addressType" minOccurs="0" maxOccurs="unbounded"
      shrex:maptoxml="true"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="addressType">
  <xs:sequence>
```

```

    <xs:element name="street" type="xs:string"/>
    <xs:element name="city" type="xs:string"/>
    <xs:element name="telephone" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

```

Here the complex element has the annotation `maptoxml=true`. Since it can occur from 0 to an indefinite number of times, it will be outlined to a separate table and the data column in that table (which, by default, gets the same name as its corresponding element) will also be nullable:

```

CREATE TABLE staff_employee (
  shrex_id INT NOT NULL,
  shrex_pid INT NOT NULL,
  id INT NOT NULL,
  name VARCHAR(100) NOT NULL,
  PRIMARY KEY(shrex_id),
  FOREIGN KEY(shrex_pid) REFERENCES staff(shrex_id)
);

CREATE TABLE staff_employee_address (
  shrex_id INT NOT NULL,
  shrex_pid INT NOT NULL,
  address XML,
  PRIMARY KEY(shrex_id),
  FOREIGN KEY(shrex_pid) REFERENCES staff_employee(shrex_id)
);

```

Example 3:

```

<xs:complexType name="telephoneType">
  <xs:sequence>
    <xs:element name="home" type="xs:string"/>
    <xs:element name="work" type="xs:string"/>
    <xs:element name="cellphone" type="xs:string" minOccurs="0"
      shrex:maptoxml="astable"/>
  </xs:sequence>
</xs:complexType>

```

Here we have the complex type `telephoneType` which has three simple elements, all of type `xs:string`. The simple element `cellphone` can only occur 0 or 1 time but since we're using the annotation `maptoxml=astable` it will be outlined to a separate table. The data column, `cellphone`, in that table will also be nullable:

```

CREATE TABLE staff_employee_address_telephone (
  shrex_id INT NOT NULL,
  shrex_pid INT NOT NULL,
  home VARCHAR(100) NOT NULL,
  work VARCHAR(100) NOT NULL,
  PRIMARY KEY(shrex_id),
  FOREIGN KEY(shrex_pid) REFERENCES staff_employee_address(shrex_id)
);

CREATE TABLE staff_employee_address_telephone_cellphone (

```

```

shrex_id INT NOT NULL,
shrex_pid INT NOT NULL,
cellphone XML,
PRIMARY KEY(shrex_id),
FOREIGN KEY(shrex_pid) REFERENCES staff_employee_address_telephone(shrex_id)
);

```

Withparenttable:

Value: true, false

Applies to: Any entity that is going to be translated to a table.

Actions: This is a way to merge a table with its parent table. Instead of a new table being created, the entity is translated to the same table as its parent, which will get all fields. If the merger results in more than one field with the same name, then ShreX will append counters to fieldnames until no conflicts remain.

Comments: It will not work if the entity can occur more than one time under its parent, and the annotation validator will catch any attempts to do so.

Examples:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:shrex="http://www.cse.ogi.edu/shrex">
  <xs:element name="cars">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="car" type="carType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="carType">
    <xs:sequence>
      <xs:element name="modelName" type="xs:string"/>
      <xs:element name="maker" type="xs:string"/>
      <xs:element name="engine" type="engineType" shrex:withparenttable="true"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="engineType">
    <xs:sequence>
      <xs:element name="engineType" type="xs:string"/>
      <xs:element name="numCylinders" type="xs:positiveInteger"/>
      <xs:element name="breakHorsePower" type="xs:positiveInteger"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>

```

Here the complex type engine has the withparenttable annotation so its inlined simple elements will be added to the table that its parent is mapped to. The resulting relational schema looks like this:

```
CREATE TABLE cars (
```

```

shrex_id INT NOT NULL,
PRIMARY KEY(shrex_id)
);

CREATE TABLE cars_car (
shrex_id INT NOT NULL,
shrex_pid INT NOT NULL,
modelName VARCHAR(100) NOT NULL,
maker VARCHAR(100) NOT NULL,
engineType VARCHAR(100) NOT NULL,
numCylinders INT NOT NULL,
breakHorsePower INT NOT NULL,
PRIMARY KEY(shrex_id),
FOREIGN KEY(shrex_pid) REFERENCES cars(shrex_id)
);

```

Ignore:

Value: true, false

Applies to: Any attribute, element or group.

Actions: This part of the XML tree is not translated at all (including any children). Any annotations on child entities are discarded without further ado.

Comments: The validator should make sure one cannot mix ignore with any other annotation for the same element and unit tests should be added to automatically test this.

Examples:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:shrex="http://www.cse.ogi.edu/shrex">
  <xs:element name="cars">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="car" type="carType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="carType">
    <xs:sequence>
      <xs:element name="modelName" type="xs:string"/>
      <xs:element name="maker" type="xs:string"/>
      <xs:element name="engine" type="xs:string" shrex:ignore="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>

```

From the schema above we get the following relational schema:

```

CREATE TABLE cars (
shrex_id INT NOT NULL,
PRIMARY KEY(shrex_id)
);

```

```
CREATE TABLE cars_car (
  shrex_id INT NOT NULL,
  shrex_pid INT NOT NULL,
  modelName VARCHAR(100) NOT NULL,
  maker VARCHAR(100) NOT NULL,
  PRIMARY KEY(shrex_id),
  FOREIGN KEY(shrex_pid) REFERENCES cars(shrex_id)
);
```

As you can see the simple element is not present in the generated relational schema.

Outline:

Value: true

Applies to: attribute or element

Actions: Use this annotation with the value “true” to create a new table for an entity that would otherwise have become a field in the current table. Any values other than “true” are ignored and this annotation cannot be used to inline a simple element (for example) that must be outlined (i.e., it can occur more than once under its parent). An entity in this case is understood to be a simple element or attribute.

Comments:

Examples:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:shrex="http://www.cse.ogi.edu/shrex">
  <xs:element name="actors">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="actor" type="actorType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="actorType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="latestMovie" type="xs:string" shrex:outline="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Here we have the simple element latestMovie that can only occur one time under its parent, so normally it would be mapped to a field in its parent table, but since we're using outline we get a new table.

```
CREATE TABLE actors (
  shrex_id INT NOT NULL,
  PRIMARY KEY(shrex_id));
```

```
CREATE TABLE actors_actor (
  shrex_id INT NOT NULL,
  shrex_pid INT NOT NULL,
  name VARCHAR(100) NOT NULL,
```

```
PRIMARY KEY(shrex_id),  
FOREIGN KEY(shrex_pid) REFERENCES actors(shrex_id));
```

```
CREATE TABLE actors_actor_latestMovie (  
  shrex_id INT NOT NULL,  
  shrex_pid INT NOT NULL,  
  latestMovie VARCHAR(100) NOT NULL,  
  PRIMARY KEY(shrex_id),  
  FOREIGN KEY(shrex_pid) REFERENCES actors_actor(shrex_id));
```