# Schema Identity Constraints

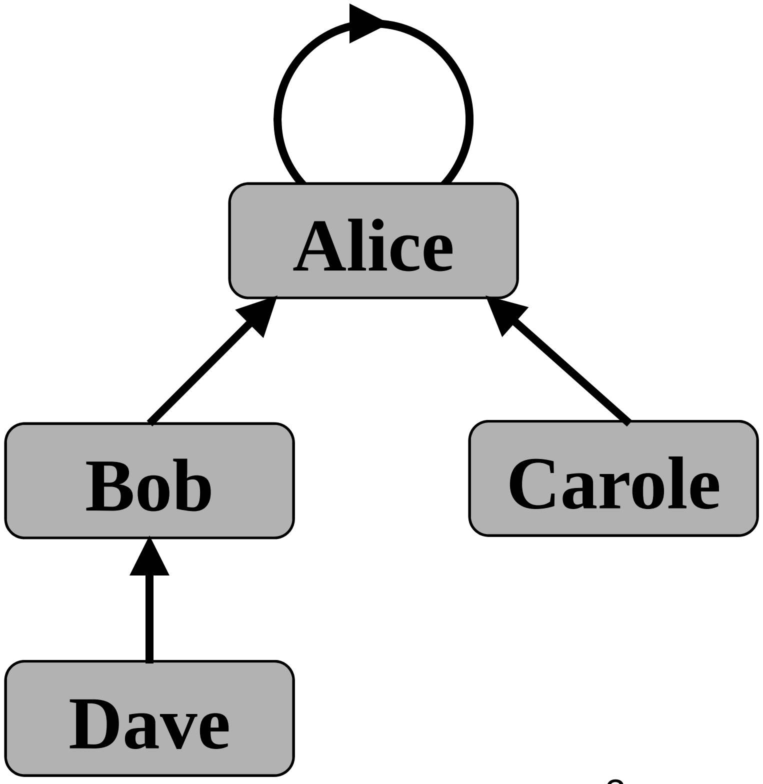
# Identity Constraints

- Recall that the attribute types **ID** and **IDREF** imply interesting constraints on *values* of those attributes:
  - Within any individual XML document, every attribute of type **ID** must be specified with a different value from *every other* attribute of type **ID**.
  - The value of any attribute of type **IDREF** must be the same as the value of an attribute of type **ID** specified *somewhere in the same document*.

# An Example

```
<agency>
   <agent name="Alice" boss="Alice"/>
   <agent name="Bob" boss="Alice"/>
   <agent name="Carole" boss="Alice"/>
   <agent name="Dave" boss="Bob"/>
</agency>
```

- Using DTDs, we assumed **name** was declared with type **ID**, and attribute **boss** was declared with type **IDREF**.



# Identity Constraints

- XML Schema enables us to define:
  - A unique constraint,
  - A key constraint, and
  - A referential integrity constraint
- These three constraints apply to a certain part of a document only and their satisfaction is checked by an XML Schema processor
- The three important concepts:
  - The context node of the constraint, scope of the constraint, and argument of the constraint
  - *Reading:*
    - *XML Schema Part 0: Primer (W3C Recommendation, 2 May 2001) http://www.w3.org/TR/xmlschema-0*
    - *XMLSchemaPart 1: Structures* http://www.w3.org/TR/xmlschema-1

# Identity Constraints

- The context node of a constraint is an element from which there is a downward path to the scope of the constraint
- The scope of a constraint defines a sequence of element instances within which the constraint has to be valid
  - A scope element  type is most often complex and multivalued
  - It is defined using a relative location path expression starting from the context node
- The argument of a constraint is a simple concept (a simple element or attribute) whose values have to obey to rules of the constraint within the scope:
  - To be unique and not null (key constraint),
  - To be unique (unique constraint), or
  - To reference an existing value of a key constraint

# KeyRef Example

```
<xsd:element name="agency">
   <xsd:complexType>
      <xsd:element ref="agent” minOccurs="0"
          maxOccurs="unbounded"/>
   </xsd:complexType>
   <xsd:key name="agentName">
      <xsd:selector xpath="agent"/>
      <xsd:field xpath="@name"/>
   </xsd:key>
   <xsd:keyref refer="agentName" name="agentBoss">
      <xsd:selector xpath="agent"/>
      <xsd:field xpath="@boss"/>
   </xsd:key>
</xsd:element>
```

# General Remarks

- The element **<xsd:key/>** defines a *key* field called **agentName**.
- The element **<xsd:keyref/>** defines a *key reference* field called **agentBoss**.
- These definitions are inside the declaration of the element **<agency/>**.
  - This implies that the scope of the uniqueness and related constraints is an individual **<agency/>** element.
  - This may or may not be the top-level element of a document.
- The fields themselves are specified by XPath expressions.

# Defining a Key

- We have the example:

```
<xsd:key name="agentName">
   <xsd:selector xpath="agent"/>
   <xsd:field xpath="@name"/>
</xsd:key>
```

  - The name of the key is **agentName**.
  - The **<xsd:selector/>** element defines the *set of nodes* labeled by this key.
    - In our case, it is the set of all **agent** elements nested directly in the **agency** element.
  - The **<xsd:field/>** element defines the field *within each labeled node* that acts as the key.
    - In our case, the **name** attribute of the node.

# Validity Constraints on Keys

- Every node identified by the XPath expression in the **<xsd:selector/>** element must have *exactly one descendant node* identified by the XPath expression in the **<xsd:field/>** element.
  - This descendant, whose value is the key field, must be an attribute or an element with *simple type*.
- No two nodes identified by **<xsd:selector/>** may have the same value for their key fields.
  - This constraint holds within the body of the scope element (the **<agency/>** element in our example).
  - But the same value of the key field is allowed on different **<agent/>** nodes inside *different* **<agency/>** elements.

# Defining a Key Reference

- We have the example:

```
<xsd:keyref refer="agentName" name="agentBoss">
    <xsd:selector xpath="agent"/>
    <xsd:field xpath="@boss"/>
</xsd:key>
```

  - The **refer** attribute is the name of the key to which we refer.
  - The **<xsd:selector/>** and **<xsd:field/>** elements identify the nodes whose values are the actual references.
    - They work in essentially the same way as in **<xsd:key/>**.
    - The two-stage approach to identifying the relevant fields is less obviously natural in this case. But it supports the generalization to multiple key fields, described later.
  - The name of the key reference is **agentBoss**—this attribute is required

# Multiple Key Fields

- A **<xsd:key/>** element can have multiple **<xsd:field/>** elements, e.g.:

```
<xsd:key name="fullName">
    <xsd:selector xpath=".//person"/>
    <xsd:field xpath="@firstName"/>
    <xsd:field xpath="@lastName"/>
</xsd:key>
```

  – For validity, this implies every **<person/>** element in scope has **firstName** and **lastName** attributes with unique pair-wise-combined values.

- A **<xsd:keyref/>** element that refers to this key must have exactly the same number of **<xsd:field/>** elements.

# schoolReport Example

```
<xsd:element name=“schoolReport”>
    <xsd:complexType>
         <xsd:sequence>
           <xsd:element name=“classes”  type=“ClassesType”/>
           <xsd:element name=“students”  type=“StudentsType”/>
         </xsd:sequence>
    </xsd:complexType>
```

# ClassesType

```
<xsd:complexType name=“ClassesType”>
   <xsd:sequence>
          <xsd:element name=“class” maxOccurs=“unbouded”>
             <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name=“student” maxOccurs=“unbounded”>
                             <xsd:complexType>
                               <xsd:attribute name=“id” type=“xsd:int”/>
                               <xsd:attribute name=“inTerm” type=“xsd:decimal”/>
                            </xsd:complexType>
                           </xsd:element>
               </xsd:sequence>
               <xsd:attribute name=“name” type=“xsd:string”/>
             </xsd:complexType>
      </xsd:element>
          </xsd:sequence>
</xsd:complexType>
```

# StudentsType

```
    <xsd:complexType name=“StudentsType”>
      <xsd:sequence>
        <xsd:element name=“student” minOccurs=“0” maxOccurs=“unbounded”>
          <xsd:complexType>
                <xsd:sequence>
                 <xsd:element name=“name” type=“xsd:string”/>
                 <xsd:element name=“surname” type=“xsd:string”/>
                </xsd:sequence>
           <xsd:attribute name=“sid” type=“xsd:int” use=“required”/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

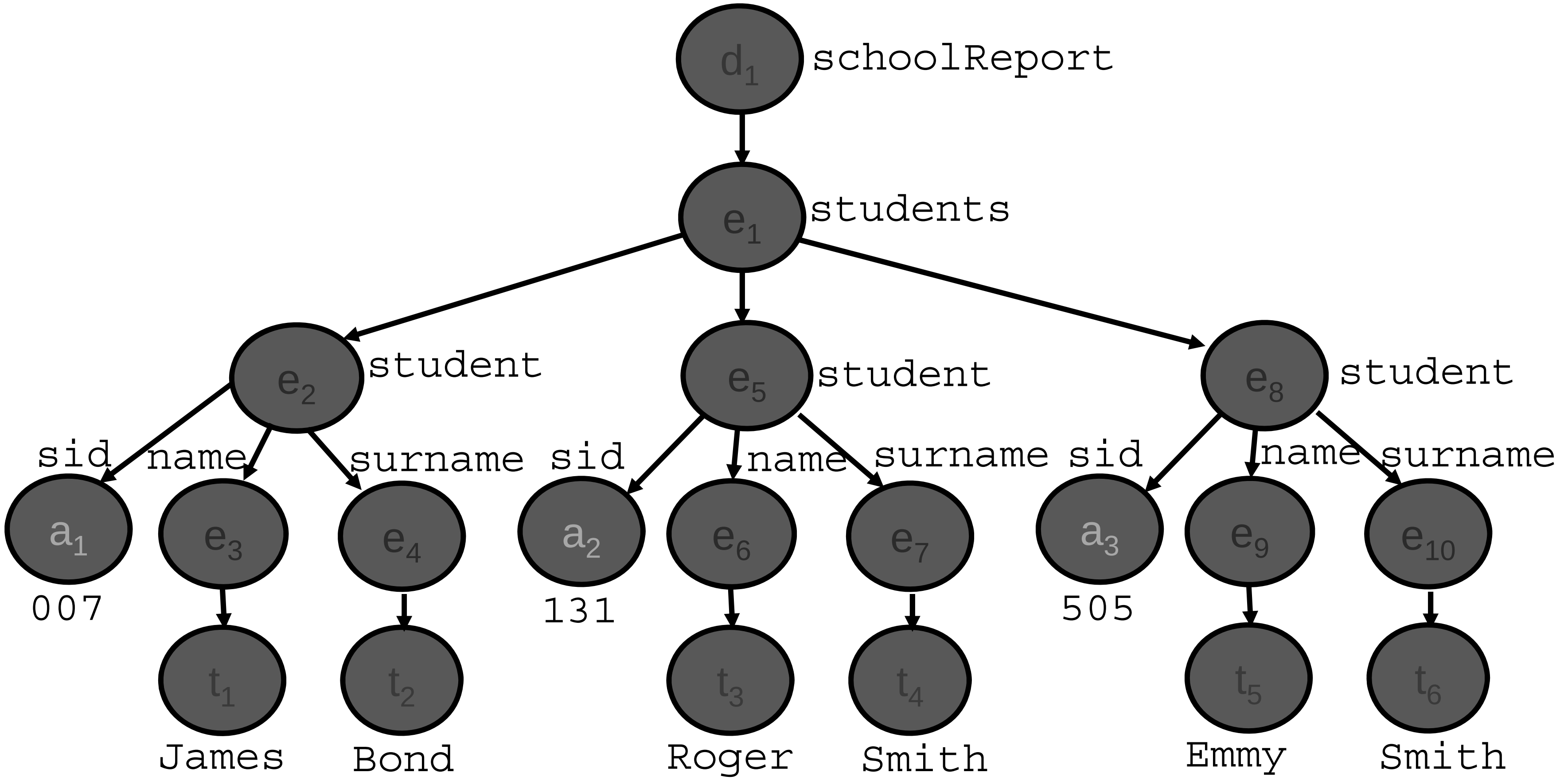
# schoolReport.xml (1 of 2)

```
<?xml version=“1.0” standalone=“yes”?>
    <schoolReport>
            <classes>
                        <class name=“comp302”>
                                    <student id=“007” inTerm=“34.75”/>
                                    <student id=“131” inTerm=“30.75”/>
                        </class>
                        <class name=“comp442”>
                                    <student id=“007” inTerm=“29.50”/>
                                    <student id=“505” inTerm=“33.00”/>
                        </class>
            </classes>
```

# schoolReport.xml

```
    <students>
            <student id=“007”>
                    <name>James</name>
                    <surname>Bond</surname>
            </student>
            <student id=“131”>
            <name>James</name>
                    <surname>Smith</surname>
            </student>
            <student id=“505”>
            <name>Emmy</name>
                    <surname>Smith</surname>
            </student>
    </students>
</schoolReport>
```

# Data Tree


d1
schoolReport
e1
students
e2
student
e5
student
e8
student
sid
name
surname
sid
name
surname
sid
name
surname
a1
e3
e4
a2
e6
e7
a3
e9
e10
007
131
505
t1
t2
t3
t4
t5
t6
James
Bond
Roger
Smith
Emmy
Smith

# Students Element

- Consider the previous subtree under the `schoolReport element`, and suppose we want to constrain `@sid` values of each `student` to be unique and not null within `students`
- So, context node is `students` ($e_1$)
- The relative location path (called `selector xpath`)

  `student`

  defines the scope and returns ($e_2$, $e_5$, $e_8$)
- The path (relative to the previously defined scope)

  `@sid`

  defines the argument (called `field`) and returns ($a_1$, $a_2$, $a_3$) whose values have to be unique

# key  and keyref  Declarations

```
     <xsd:key name=“stPrimKey”>
        <xsd:selector xpath=“students/student”/>
        <xsd:field xpath=“@id”/>
     </xsd:key>

     <xsd:keyref name=“refStudId” refer=“stPrimKey”>
        <xsd:selector path=“classes/class/student”/>
        <xsd:field xpath=“@id”/>
     </xsd:keyref>
</xsd:element>
```

# Identity Constraint

- An identity constraint is defined using the following three elements:
  - `unique | key | keyref` (to declare the kind of a constraint and to name it),
  - `selector` (to define the scope of the constraint), and
  - `field` (to designate a simple concept whose values have to be checked)
  - A `field` can be either an element having simple type values, or an attribute
  - If the constraint type is `unique` or `keyref, field` value can be declared to be optional or `nilable`
- Syntax:

```
<xsd:constraint_type name=“constraint_name”>
    <xsd:selector xpath=“e_1/…/e_n”/>
    <xsd:field xpath=“e_{n+1}/…/field_name”/>
</xsd:constraint_type>
```

# Simple Unique Constraint (Attribute)

- Suppose the next declaration is specified within the declaration of the `classes` element of the `SchoolReport.xsd`

```
<xsd:unique name=“clsName”>
    <xsd:selector xpath=“class”/>
    <xsd:field xpath=“@name”/>
</xsd:unique>
```

- Then the `@name` value of each `class` in the `classes` have to be unique
- The context node of the `xsd:selector`’s `xpath` is `classes`
- The sequence of context nodes of the `xsd:field`’s `xpath` are `class` nodes ($e_3$, $e_8$)
- So, `xsd:field`‘s `xpath` specifies that there should be a unique value of the `@name` attribute associated with each `class` element in ($e_3$, $e_8$) (this also implies ($a_1$, $a_4$) should be unique)

# Composite Unique Constraint (Element)

- Consider the `SchoolReport.xml` and suppose the next declaration is specified within the content model of the `students` element

```
<xsd:unique name="stNameAndSurname">
    <xsd:selector xpath="student"/>
    <xsd:field xpath="name"/>
    <xsd:field xpath="surname"/>
</xsd:unique>
```

- Then the `string` values of each combination of `name` and `surname` elements in the `students/student` have to be unique
- The context node of the `xsd:selector`'s `xpath` is `students` ($e_{13}$)
- The context nodes of the `xsd:field`'s `xpath` are `student` nodes ($e_{14}$ , $e_{17}$, $e_{20}$ )

# Defining Keys & Their References

- The syntax for the key constraint is very similar to the syntax for defining the unique constraint
- The key declaration on a field means that (within the scope specified) field values:
  - Must be unique, and
  - Cannot be set to `nil`, or omitted
- A key enables defining a referential integrity constraint within the given scope

# Key and Referential Integrity Example

- To insure that each student id in `classes/class` belongs to a real student in `students`, a key is defined within the declaration of the `schoolReport` element

```
<xsd:key name=“stPrimKey”>
    <xsd:selector xpath=“students/student”/>
    <xsd:field xpath=“@id”/>
</xsd:key>
```

and it is referenced by

```
<xsd:keyref name=“refStudId” refer=“stPrimKey”>
    <xsd:selector xpath=“classes/class/student”/>
    <xsd:field xpath=“@id”/>
</xsd:keyref>
```

that is also defined within the `schoolReport` element

# Placement of `key` and `keyref` Constraints

- The `key` constraint has to be "visible" to the `keyref` constraint
  - i.e. there has to exist a non upwards path from the `keyref` definition to the `key` definition
- That means we can place
  - Both `key` and `keyref` within the `schoolReport` element, or
  - The `key` constraint within the `students` element and `keyref` within the `schoolReport` element
- But we can't place:
  - The `key` constraint within the `schoolReport` element and the `keyref` constraint within the `classes` element, or
  - The `key` constraint within the `students` element and `keyref` within the `classes` element