SCHEMAS FOR XML DATA

Schemas for XML

- In XML you can define your own markup languages
  - via, external grammars, aka types, aka schemas

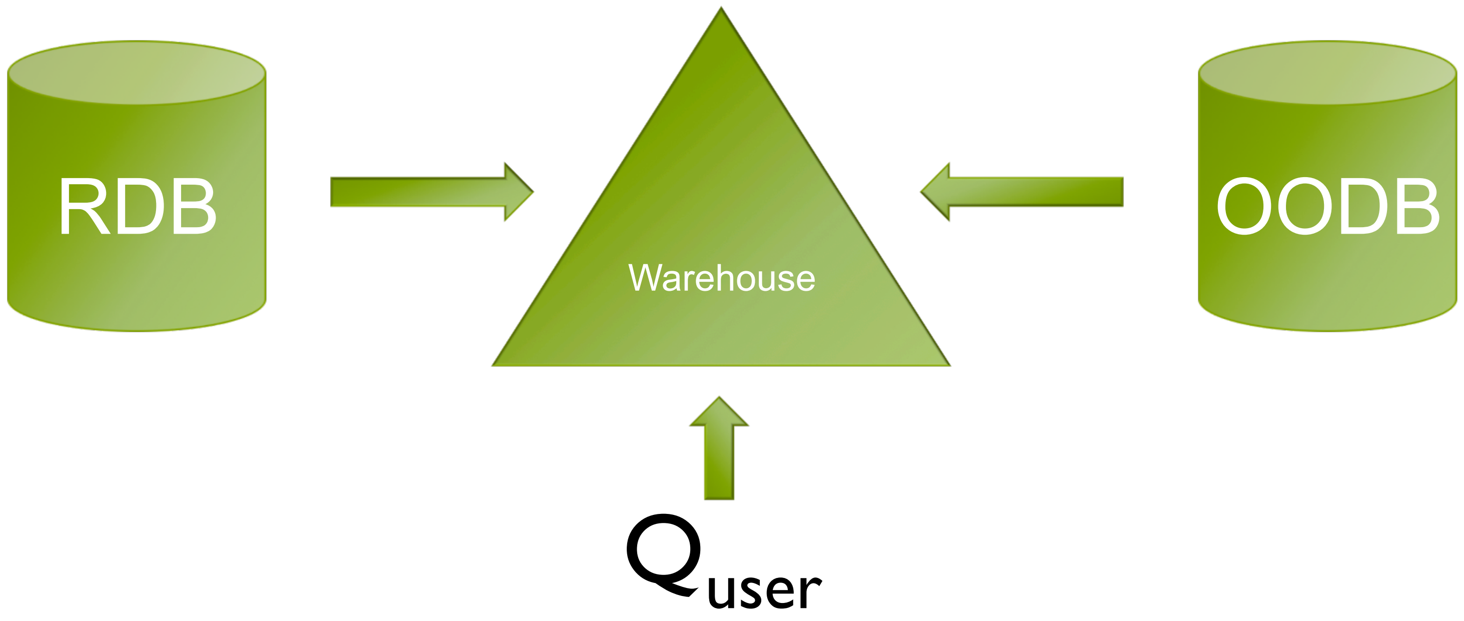
- Two notions arises. A document is :
  - Well-formed : tags match properly and attributes are “ok”
  - Valid : there is a schema and the document matches it

Wait...

- ...didn’t we just say that one of the strong points of XML is that it is a schema-less format ?
- …why falling back to schemas ?

Data exchange and integration

- Impossible if a schema between peers is not agreed !



Why schemas do help

- Interoperability/reliability
  - specify required, optional, default values
- Consistency
  - ensure updates or generated output is coherent
- Efficiency
  - use to organize storage ; for query optimization

Schemas

- Many schema languages/formalisms have been proposeed
  - DTD (XML 1.0)
  - XML Schema (W3C)
  - Relax/NG (OASIS), DSD, Schematron, ...
  - Regular expression types (XDuce, XQuery)

- Guess what : all of these are based on regular expressions !

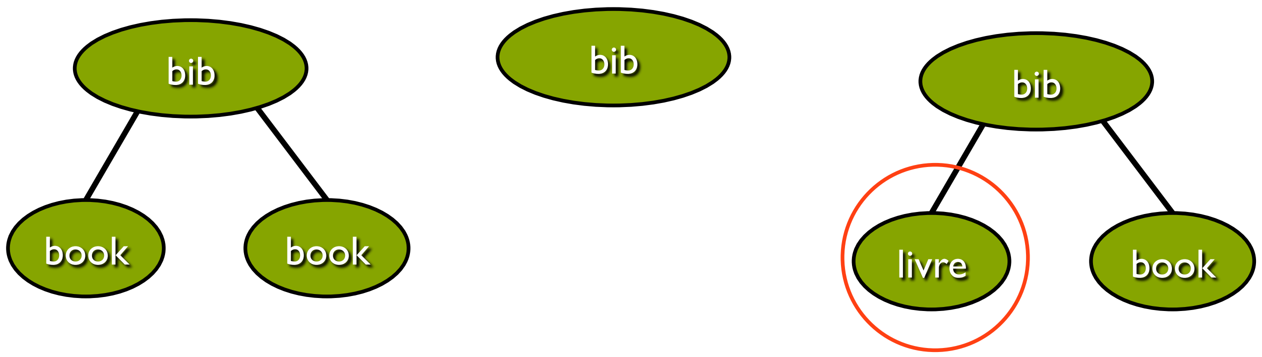
DTDs

DTD : Document Type Definition

- This schema language allows you to understand the ideas at the basis of all other proposal (XSD, RelaxNG, ...)
- Its main components are the definitions of elements and attributes
  - basic features which make up 90% of the application needs

Element declaration

```
<!ELEMENT bib (book*)>
```



Element declaration

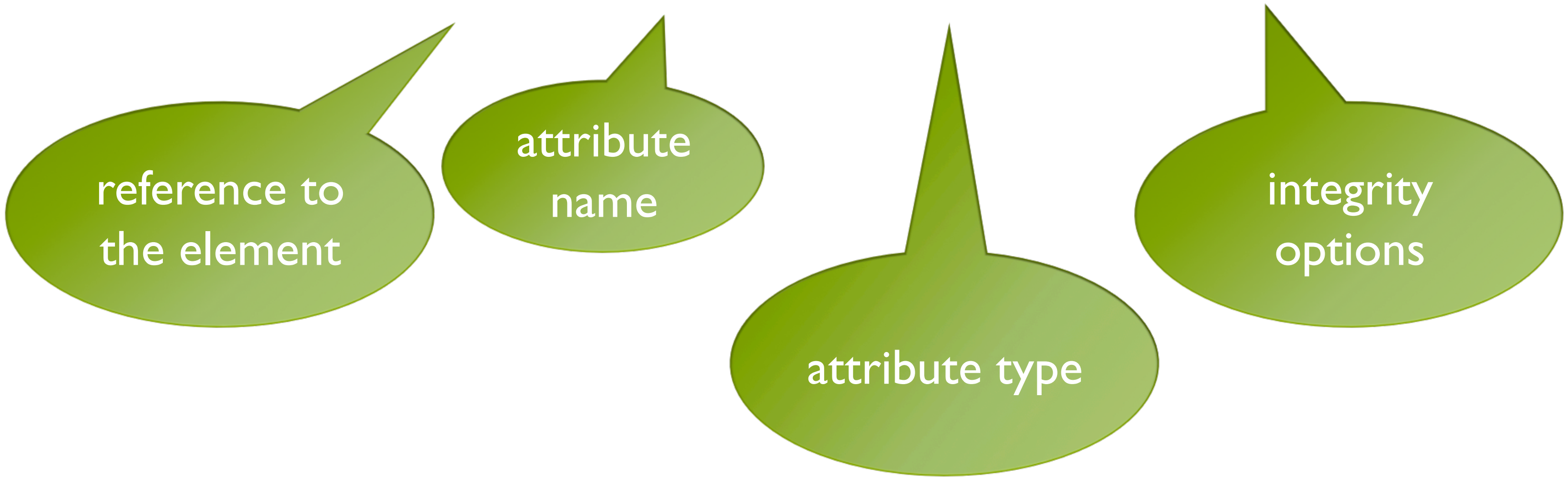
```
<!ELEMENT bib (book*)>
```

- content usually a regular expression over element names
- also allowed: `ANY`, `EMPTY`, `PCDATA` (for text)

- The declaration ANY allows one to use any **declared** type.

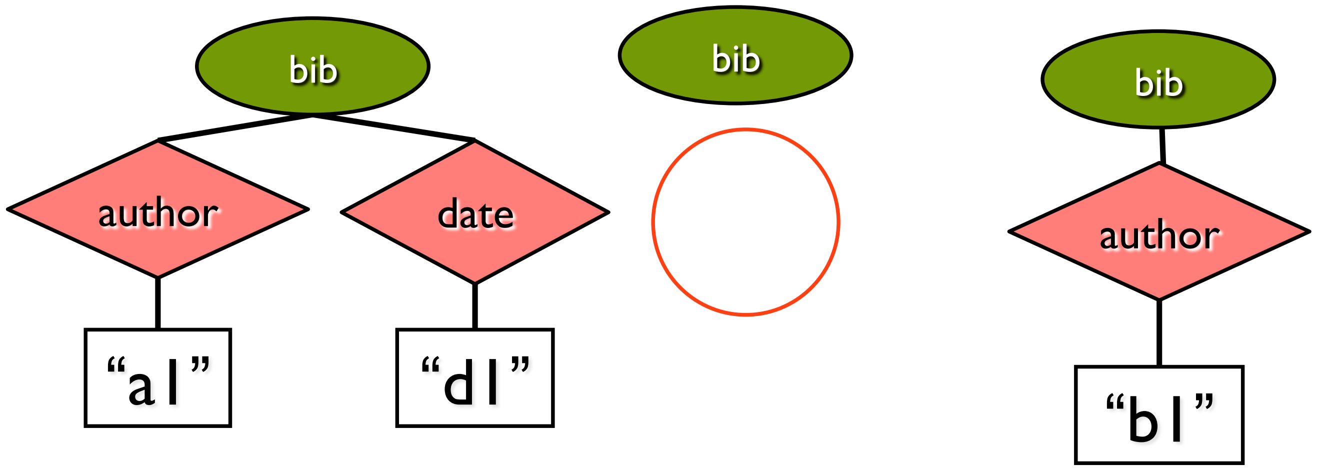
Attribute declaration

```
<!ATTLIST bib author CDATA #REQUIRED>
```



Attribute declaration

```
<!ATTLIST bib author CDATA #REQUIRED>

<!ATTLIST bib date   CDATA #IMPLIED>
```



Important :

Always make reference to an element (e.g. `bib`)

```
<!ATTLIST bib author CDATA #REQUIRED>

<!ATTLIST bib date   CDATA #IMPLIED>
```

Attribute declaration

Attributes can also be `#FIXED` value

```
<!ATTLIST book owner  #FIXED “Bob” >
```

Attributes can also be of fixed domain (enumeration)

```
<!ATTLIST book category (comic|fantasy) >
```

Remember this ?

- Use attributes for IDs and Keys !

```
<!ATTLIST person pid ID #REQUIRED>
```

- **ID** imposes that the value of the attribute value must be unique within document

Remember this ?

- Use attributes for IDs and Keys !

```
<!ATTLIST person pid IDREF #REQUIRED>
```

- **`IDREF`** imposes that attribute value must appear somewhere in the document as an **`ID`**

ID / **IDREF**

```
<!ATTLIST person pid    ID    #REQUIRED>
<!ATTLIST person friend IDREF #IMPLIED>
```

```
<person pid = “p1”   friend=“p2”>
      <name> Barak Obama </name>
</person>


<person pid = “p2”   friend=“p1”>
      <name> Bill Clinton </name>
</person>
```

ID / IDREF

```
<!ATTLIST person  pid     ID     #REQUIRED>
<!ATTLIST person  friend IDREF #IMPLIED>
```

```
<person pid = “
      <name>
</person>
```

ID attributes must start with a letter!

```
<person pid = “p2”   friend=“p1”>
      <name> Bill Clinton </name>
</person>
```

ID / IDREF

```
<!ATTLIST person  pid     ID     #REQUIRED>
<!ATTLIST person  friend IDREF  #REQUIRED>
```

```
<person pid = “p1”   friend=“p2”>
      <name> Barak Obama </name>
</person>


<person pid = “p2”   friend=“p1”>
      <name> Bill Clinton </name>
</person>
```

ID / **IDREF**

```
<!ATTLIST person  pid      ID      #REQUIRED>
<!ATTLIST person  friend IDREF   #REQUIRED>
```

```
<person pid = “p1”                        >
        <name> Barak Obam     /name>
</person>

<person pid = “p2”                     p1”>
        <name> Bill Clinton </name>
</person>
```

missing attribute

ID / IDREF

```
<!ATTLIST person  pid     ID     #IMPLIED>
<!ATTLIST person  friend IDREF  #REQUIRED>
```

```
  <person              friend="p2">
        <name> Barak Obama </name>
  </person>
```

missing reference

```
  <person              friend="p1">
        <name> Bill Clinton </name>
  </person>
```

missing reference

Quiz

```
<!ATTLIST person pid    ID     #IMPLIED>
<!ATTLIST person friend IDREF  #IMPLIED>
```

```
  <person >
          <name> Barak Obama </name>
  </person>



  <person >
          <name> Bill Clinton </name>
  </person>
```

DTD example

```
<!DOCTYPE bib[
  <!ELEMENT bib  ( book* )>
  <!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>
  <!ATTLIST book  year CDATA  #REQUIRED >
  <!ELEMENT author  (last, first )>
  <!ELEMENT editor  (last, first )>
  <!ELEMENT title  (#PCDATA )>
  <!ELEMENT last  (#PCDATA )>                <!ELEMENT first  (#PCDATA )>
  <!ELEMENT publisher  (#PCDATA )>  <!ELEMENT price  (#PCDATA )>
]>
```

Quiz

```
<!ELEMENT table (row*)>

<!ATTLIST table title CDATA #REQUIRED>

<!ELEMENT row (A,(B|C))>

<!ELEMENT row (A|C) >

<!ELEMENT A (#PCDATA)>

<!ELEMENT B (#PCDATA)>

<!ELEMENT C (#PCDATA)>
```

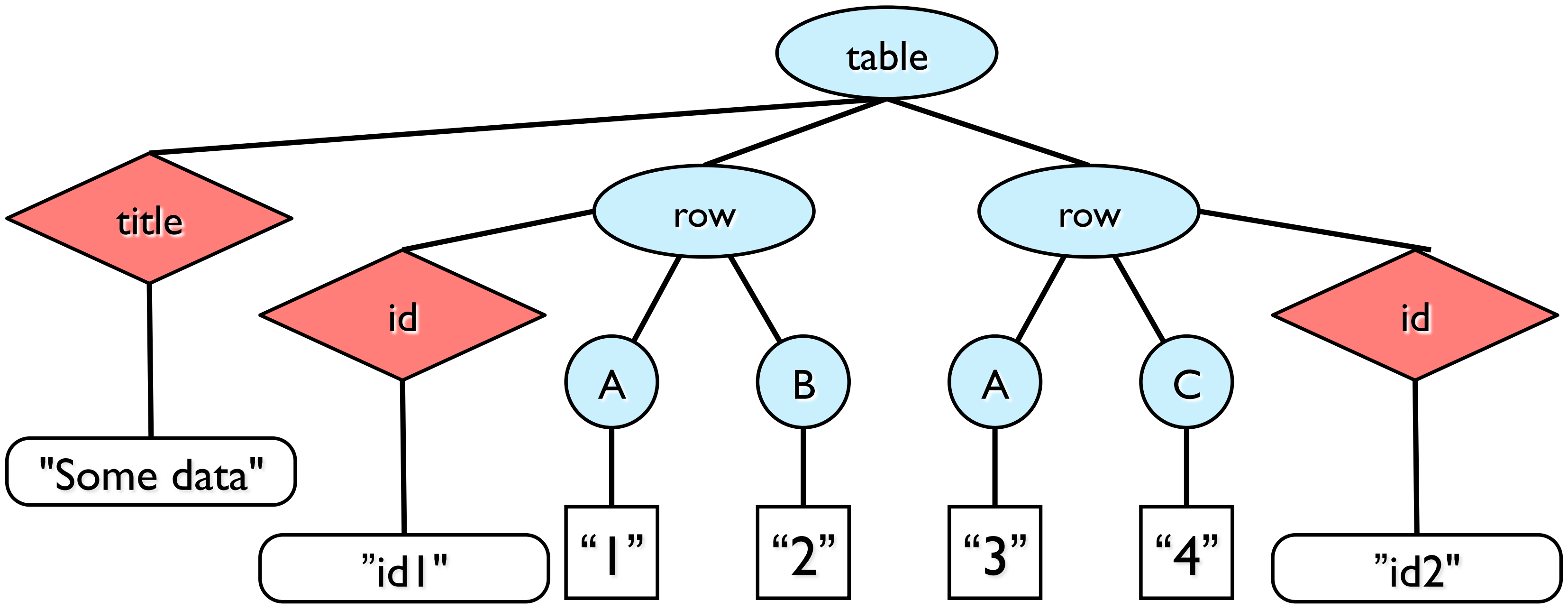
Quiz : find the error(s)

```
<!ELEMENT table (row*)>

<!ATTLIST table title CDATA #REQUIRED>

<!ELEMENT row (A,(B|C))>

<!ELEMENT row (A|C) >

<!ELEMENT A (#PCDATA)>

<!ELEMENT B (#PCDATA)>

<!ELEMENT C (#PCDATA)>
```

cannot define twice the same tag `row` !

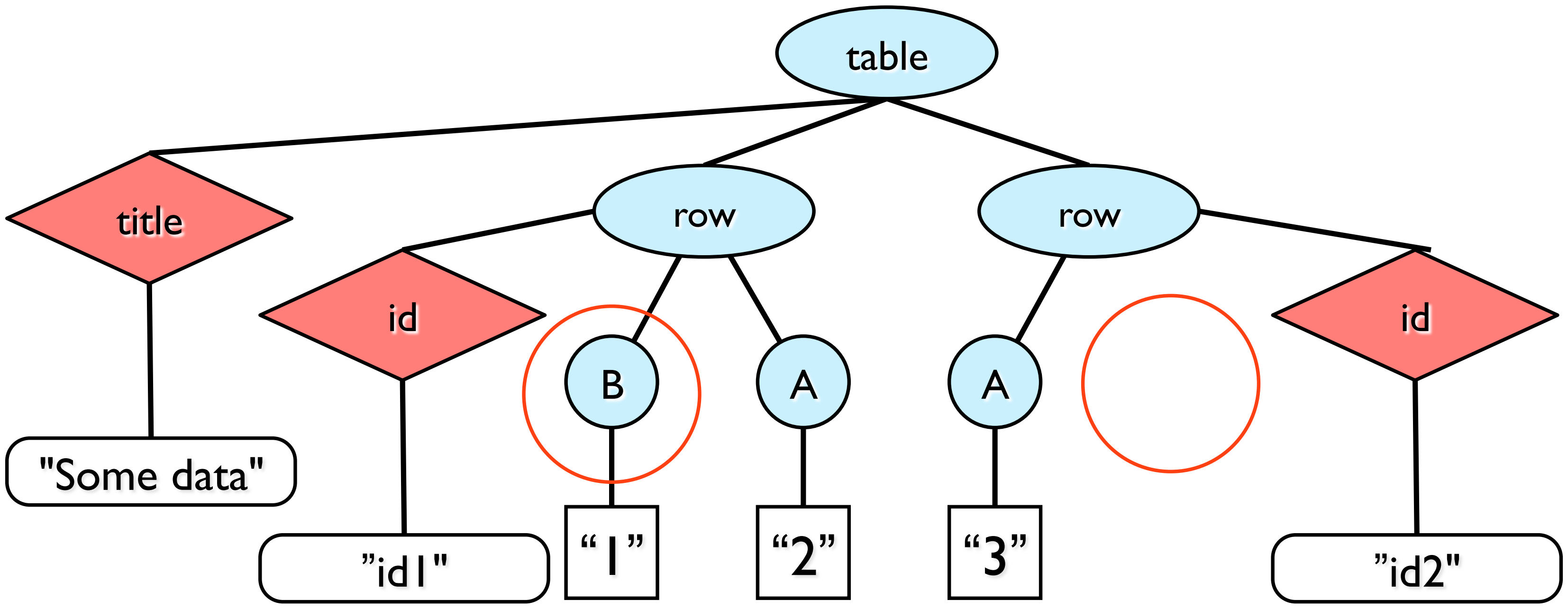
Quiz1

```
<!ELEMENT table (row*)>
<!ATTLIST table title CDATA #REQUIRED>
<!ELEMENT row (A,(B|C))>
<!ATTLIST row id ID #REQUIRED>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```



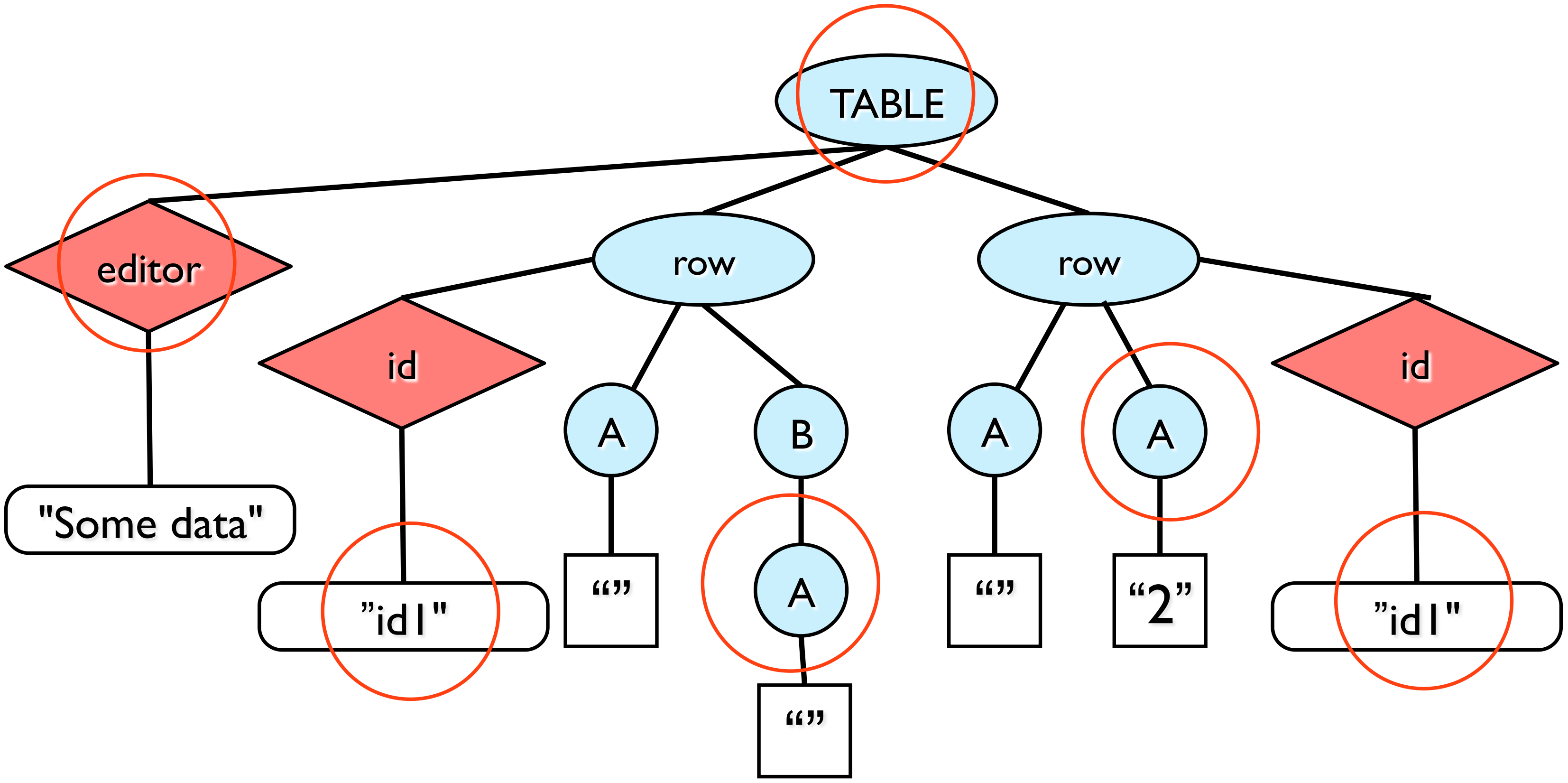
Quiz2

```
<!ELEMENT table (row*)>
<!ATTLIST table title CDATA #REQUIRED>
<!ELEMENT row (A,(B|C))>
<!ATTLIST row id ID #REQUIRED>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```



Quiz3

```
<!ELEMENT table (row*)>
<!ATTLIST table title CDATA #REQUIRED>
<!ELEMENT row (A,(B|C))>
<!ATTLIST row id ID #REQUIRED>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```



Recursive Elements

DTD rules can be recursive

- `node → (node,node)?`

Recursion increases complexity of DTD

- This leads to documents of unbounded depth
- Some element types might not have any finite matching trees
- but this is easy to detect (look for unguarded cycles)
  - `silly → (silly, silly)`

Limitations of DTDs

- Can't constrain text / attribute content (except in very limited ways)
- Can't specialize tag, e.g. "name", in different ways/contexts (while XSD allows this)
  - eg., name of an author, name of a company : need to create two tags
- Element, attribute content are context insensitive
- ID/IDRef satisfy weak integrity conditions

Quiz

Give a document valid for this DTD, if it exists; otherwise explain why it does not exists.

```
<!ELEMENT X (Y)>
<!ELEMENT Y (A,B,X)>
<!ELEMENT A EMPTY>
<!ELEMENT B (A,B)*>
```

Give a DTD for which only the following XML tree is valid (=no other XML tree is valid!).

```
<A>
  <B/> <B/>   <B/>
<A/>
```

Give a document valid for this DTD, if it exists; otherwise explain why it does not exists.

```
<!ELEMENT Y (A)>
<!ELEMENT A EMPTY>
<!ELEMENT A (A,B)*>
```

XML and DTD together

Coupled

```
<?xml version="1.0"?>


<!DOCTYPE bib [

 <!ELEMENT bib book*>
 ...

]>

<bib> </bib>
```

Decoupled

bib.dtd

```
~~<!DOCTYPE bib [~~

 <!ELEMENT bib book*>
 ...

~~]>~~
```

```
<?xml version="1.0">

<!DOCTYPE bib SYSTEM "bib.dtd">

<bib> </bib>
```