Document Type Definition (DTD) Reference

Schemas and schema languages

A schema is a definition of the structures of XML documents

A schema language is a formal language for expressing schemas

XML document is valid if its structures are consistent with the structures defined in its schema

Schema processing: given an XML document and a schema, a schema processor checks for validity, i.e. that the document conforms to the schema requirements

If the document is valid then its normalized version is created: default attributes and elements are inserted

Schemas are similar to grammars for programming languages

Proposals for schema languages

W3C proposals:

- DTD
- XML-Data, January 1998
- DCD (Document Content Description), July 1998
- DDML (Document Definition Markup Language), Jan 1999
- SOX (Schema for Object-oriented XML), July 1999
- XML Schema

Non-W3C proposals:

- Assertion Grammars by Dave Raggett
- Schematron by Rick Jellife
- TREX (Tree Regular Expressions for XML) by James Clark
- Examplotron by Eric van der Vlist
- RELAX by Makoto Murara / RELAX NG by Murata and Clark
- DSD (Document Structure Description)

Document Type Definition

Document Type Definition (DTD) defines the structures of XML documents

Document Type Definition (DTD) uses the notation of regular expressions (see Appendix) to define XML documents

Document Type Definition (DTD) consists of the following components: name of root element, element declarations, attribute lists, entity declarations, and conditional sections

Document Type Definition

Components of DTD:

Name of root element and document type declarations

```
<!DOCTYPE root-element [ doctype-declaration... ]>
```

Element declaration

```
<!ELEMENT element-name content-model>
```

Attribute list

```
<!ATTLIST element-name attr-name attr-type attr-default ...>
```

Entity declaration

```
<!ENTITY entity-name "entity-value">
```

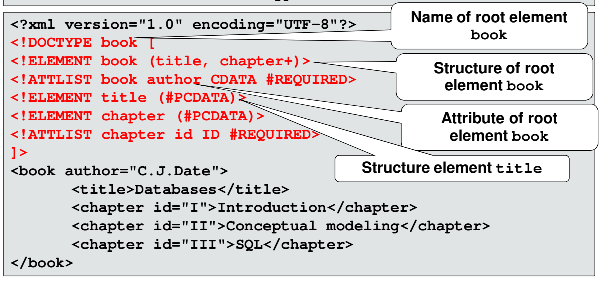
Conditional sections

```
<![switch[DTD text]] >
```

Name of root element and document type declaration

Internal DTD declaration

```
<!DOCTYPE root-element [ doctype-declaration... ]>
```



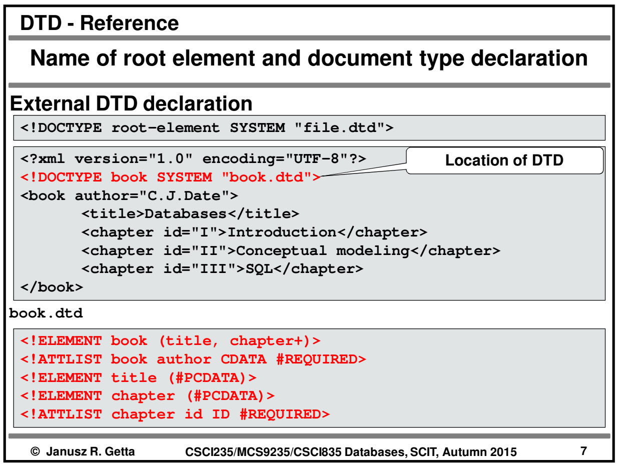
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
<!ELEMENT book (title, chapter+)>
<!ATTLIST book author CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT chapter (#PCDATA)>
<!ATTLIST chapter id ID #REQUIRED>
]>
<book author="C.J.Date">
       <title>Databases</title>
       <chapter id="I">Introduction</chapter>
       <chapter id="II">Conceptual modeling</chapter>
       <chapter id="III">SQL</chapter>
</book>
```

Name of root element book

Structure of root element book

Attribute of root element book

Structure element title



Name of root element and document type declaration

External DTD declaration

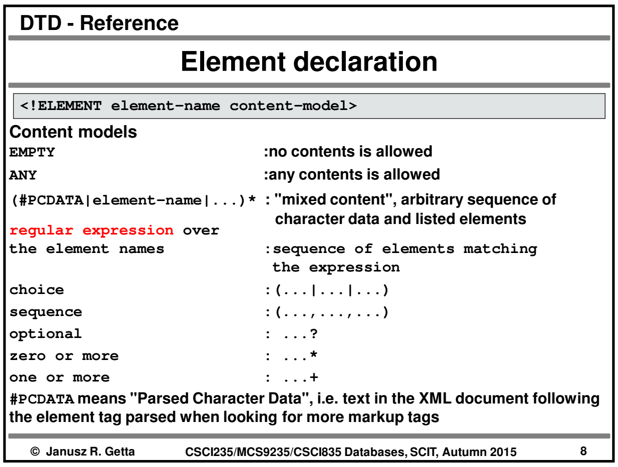
```
<!DOCTYPE root-element SYSTEM "file.dtd">
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM "book.dtd">
<book author="C.J.Date">
       <title>Databases</title>
       <chapter id="I">Introduction</chapter>
       <chapter id="II">Conceptual modeling</chapter>
       <chapter id="III">SQL</chapter>
</book>
```

Location of DTD

book.dtd

```
<!ELEMENT book (title, chapter+)>
<!ATTLIST book author CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT chapter (#PCDATA)>
<!ATTLIST chapter id ID #REQUIRED>
```



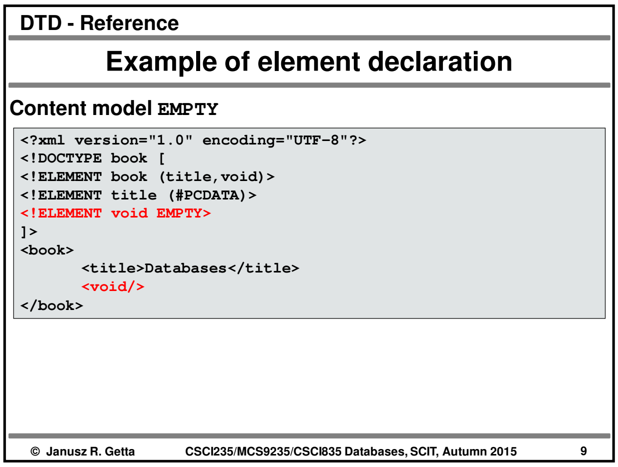
Element declaration

```
<!ELEMENT element-name content-model>
```

Content models

`EMPTY`	:no contents is allowed
`ANY`	:any contents is allowed
`(#PCDATA|element-name|...)*`	: "mixed content", arbitrary sequence of character data and listed elements
`regular expression over the element names`	:sequence of elements matching the expression
`choice`	`:(...|...|...)`
`sequence`	`:(...,...,...)`
`optional`	`: ...?`
`zero or more`	`: ...*`
`one or more`	`: ...+`

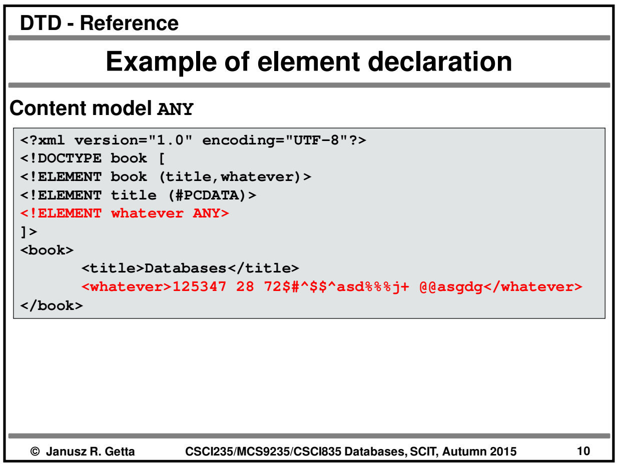
`#PCDATA` means "Parsed Character Data", i.e. text in the XML document following the element tag parsed when looking for more markup tags



Example of element declaration

Content model `EMPTY`

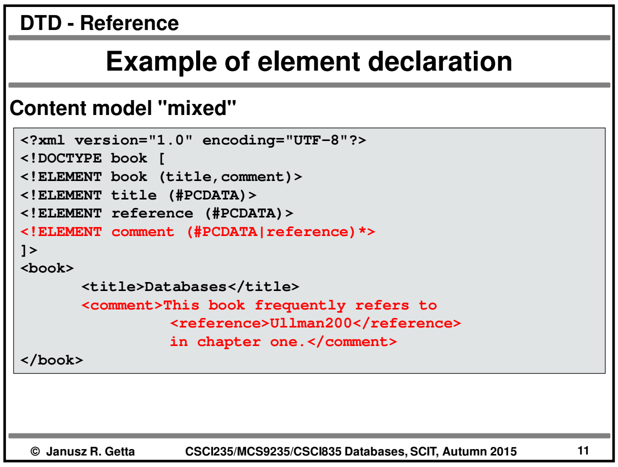
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
<!ELEMENT book (title,void)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT void EMPTY>
]>
<book>
       <title>Databases</title>
       <void/>
</book>
```



Example of element declaration

Content model `ANY`

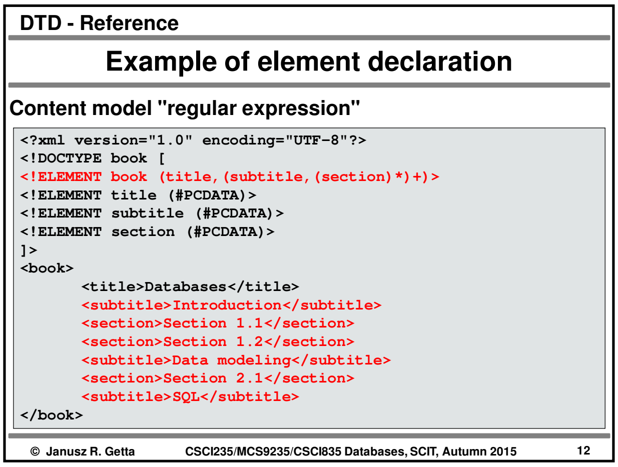
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
<!ELEMENT book (title,whatever)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT whatever ANY>
]>
<book>
       <title>Databases</title>
       <whatever>125347 28 72$#^$$^asd%%%j+ @@asgdg</whatever>
</book>
```



Example of element declaration

Content model "mixed"

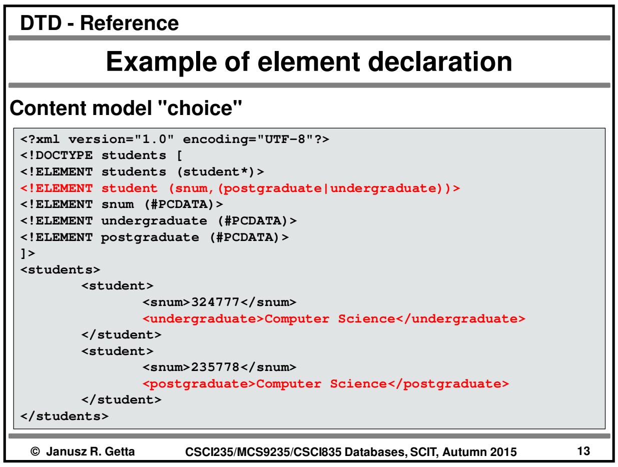
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
<!ELEMENT book (title,comment)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT reference (#PCDATA)>
<!ELEMENT comment (#PCDATA|reference)*>
]>
<book>
       <title>Databases</title>
       <comment>This book frequently refers to
                <reference>Ullman200</reference>
                in chapter one.</comment>
</book>
```



Example of element declaration

Content model "regular expression"

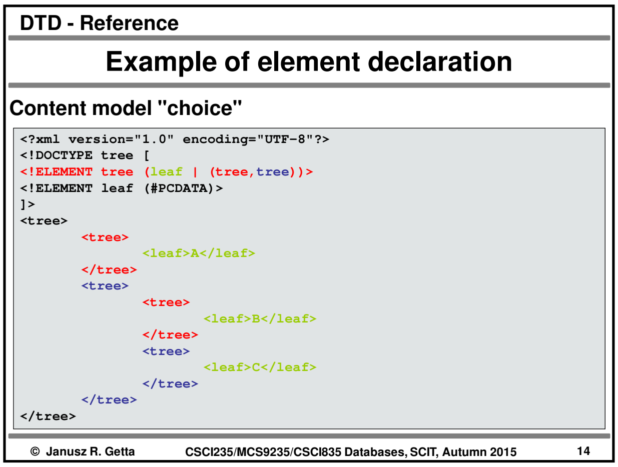
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
<!ELEMENT book (title,(subtitle,(section)*)+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT subtitle (#PCDATA)>
<!ELEMENT section (#PCDATA)>
]>
<book>
       <title>Databases</title>
       <subtitle>Introduction</subtitle>
       <section>Section 1.1</section>
       <section>Section 1.2</section>
       <subtitle>Data modeling</subtitle>
       <section>Section 2.1</section>
       <subtitle>SQL</subtitle>
</book>
```



Example of element declaration

Content model "choice"

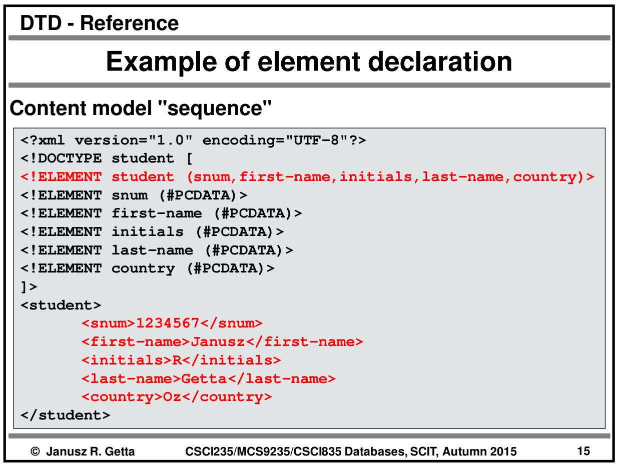
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE students [
<!ELEMENT students (student*)>
<!ELEMENT student (snum,(postgraduate|undergraduate))>
<!ELEMENT snum (#PCDATA)>
<!ELEMENT undergraduate (#PCDATA)>
<!ELEMENT postgraduate (#PCDATA)>
]>
<students>
        <student>
                <snum>324777</snum>
                <undergraduate>Computer Science</undergraduate>
        </student>
        <student>
                <snum>235778</snum>
                <postgraduate>Computer Science</postgraduate>
        </student>
</students>
```



Example of element declaration

Content model "choice"

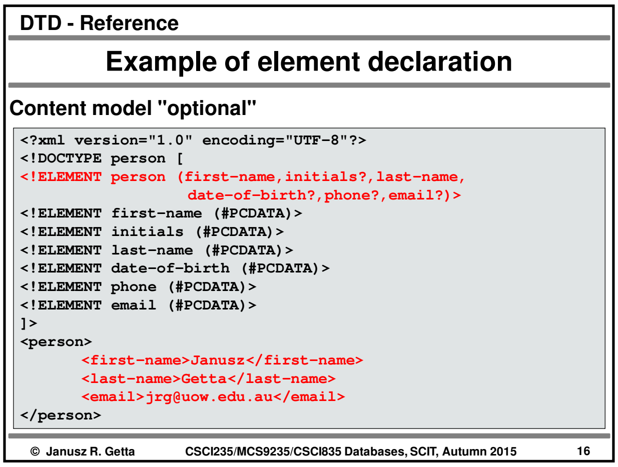
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tree [
<!ELEMENT tree (leaf | (tree,tree))>
<!ELEMENT leaf (#PCDATA)>
]>
<tree>
        <tree>
                <leaf>A</leaf>
        </tree>
        <tree>
                <tree>
                        <leaf>B</leaf>
                </tree>
                <tree>
                        <leaf>C</leaf>
                </tree>
        </tree>
</tree>
```



Example of element declaration

Content model "sequence"

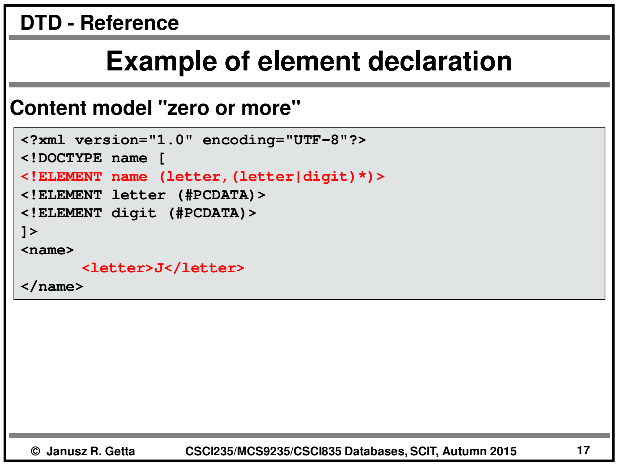
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student [
<!ELEMENT student (snum,first-name,initials,last-name,country)>
<!ELEMENT snum (#PCDATA)>
<!ELEMENT first-name (#PCDATA)>
<!ELEMENT initials (#PCDATA)>
<!ELEMENT last-name (#PCDATA)>
<!ELEMENT country (#PCDATA)>
]>
<student>
        <snum>1234567</snum>
        <first-name>Janusz</first-name>
        <initials>R</initials>
        <last-name>Getta</last-name>
        <country>Oz</country>
</student>
```



Example of element declaration

Content model "optional"

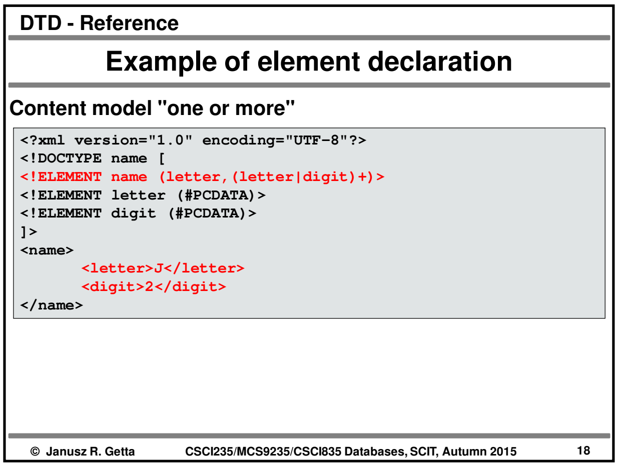
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE person [
<!ELEMENT person (first-name,initials?,last-name,
                  date-of-birth?,phone?,email?)>
<!ELEMENT first-name (#PCDATA)>
<!ELEMENT initials (#PCDATA)>
<!ELEMENT last-name (#PCDATA)>
<!ELEMENT date-of-birth (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT email (#PCDATA)>
]>
<person>
        <first-name>Janusz</first-name>
        <last-name>Getta</last-name>
        <email>jrg@uow.edu.au</email>
</person>
```



Example of element declaration

Content model "zero or more"

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE name [
<!ELEMENT name (letter,(letter|digit)*)>
<!ELEMENT letter (#PCDATA)>
<!ELEMENT digit (#PCDATA)>
]>
<name>
        <letter>J</letter>
</name>
```



Example of element declaration

Content model "one or more"

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE name [
<!ELEMENT name (letter,(letter|digit)+)>
<!ELEMENT letter (#PCDATA)>
<!ELEMENT digit (#PCDATA)>
]>
<name>
        <letter>J</letter>
        <digit>2</digit>
</name>
```

Attribute list

```
<!ATTLIST element-name attr-name attr-type attr-default-value >
```

Attribute types

`CDATA`	:any value is allowed (default)
`(value|...)`	:enumeration of allowed values
`ID`	:a value is a unique identifier
`IDREF`	:a value is an ID of another element (a reference to element)
`IDREFS`	:a value is a list of IDs of another element (the references to element)
`NMTOKEN`	:a value is a valid XML name
`NMTOKENS`	:a value is a list of valid XML names
`ENTITY`	:a value is an entity
`ENTITIES`	:a value is a list of entities
`NOTATION`	:a value is a name of a notation
`xml:`	:a value is a predefined XML value

Attribute list

```
<!ATTLIST element-name attr-name attr-type attr-default-value >
```

Attribute default values

`value`	:a given value is a default value
`#REQUIRED`	:a value is required
`#IMPLIED`	:a value is not required
`#FIXED value`	:a given value is fixed

Example of attribute list

Attribute type `CDATA`, default `#REQUIRED`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
<!ELEMENT book (title)>
<!ATTLIST book author CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
]>
<book author="C.J.Date">
       <title>Databases</title>
</book>
```

Example of attribute list

Attribute type `ID`, default `#REQUIRED`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
<!ELEMENT book (title,chapter+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT chapter (#PCDATA)>
<!ATTLIST chapter id ID #REQUIRED>
]>
<book>
       <title>Databases</title>
       <chapter id="I">Introduction</chapter>
       <chapter id="II">Conceptual modeling</chapter>
       <chapter id="III">SQL</chapter>
</book>
```

Example of attribute list

Attribute type `(value|...)`, default `#IMPLIED`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE degree [
<!ELEMENT degree (subject+)>
<!ELEMENT subject (#PCDATA)>
<!ATTLIST subject credits (6 | 12) #IMPLIED>
]>
<degree>
       <subject credits="6">Databases</subject>
       <subject>Java for managers</subject>
       <subject credits="12">Software Project</subject>
</degree>
```

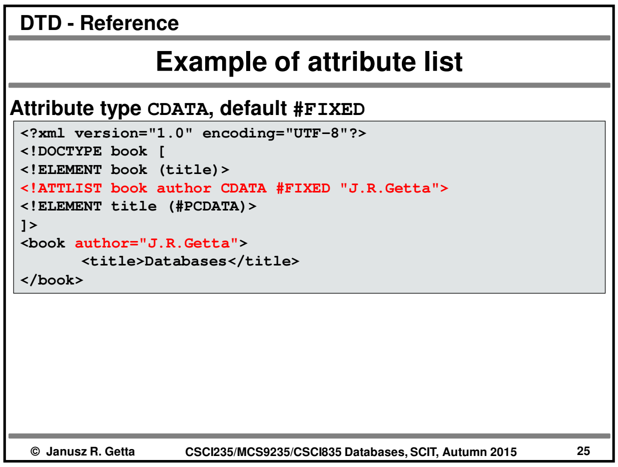
Example of attribute list

Attribute type `CDATA`, default `#FIXED`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
<!ELEMENT book (title)>
<!ATTLIST book author CDATA #FIXED "J.R.Getta">
<!ELEMENT title (#PCDATA)>
]>
<book author="C.J.Date">
       <title>Databases</title>
</book>
```

A document above does not validate !

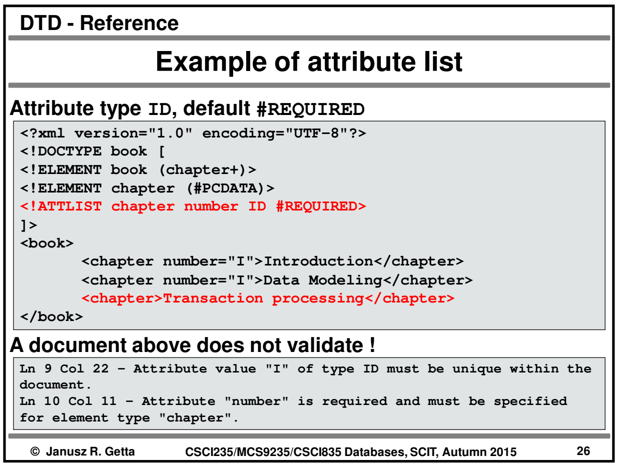
```
Ln 7 Col 25 - Attribute "author" with value "C.J.Date" must
have a value of "J.R.Getta".
```



Example of attribute list

Attribute type `CDATA`, default `#FIXED`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
<!ELEMENT book (title)>
<!ATTLIST book author CDATA #FIXED "J.R.Getta">
<!ELEMENT title (#PCDATA)>
]>
<book author="J.R.Getta">
       <title>Databases</title>
</book>
```



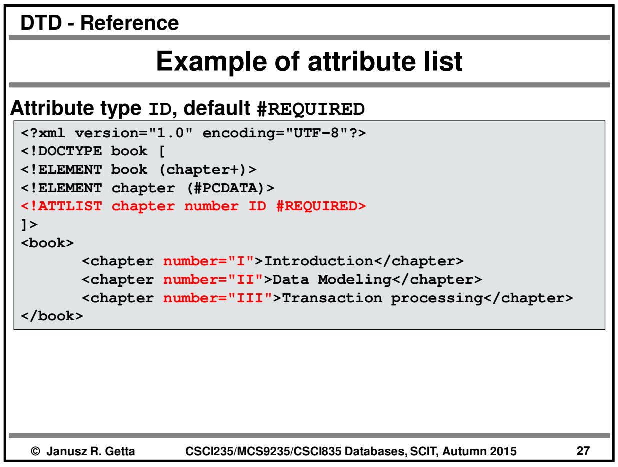
Example of attribute list

Attribute type `ID`, default `#REQUIRED`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
<!ELEMENT book (chapter+)>
<!ELEMENT chapter (#PCDATA)>
<!ATTLIST chapter number ID #REQUIRED>
]>
<book>
       <chapter number="I">Introduction</chapter>
       <chapter number="I">Data Modeling</chapter>
       <chapter>Transaction processing</chapter>
</book>
```

A document above does not validate !

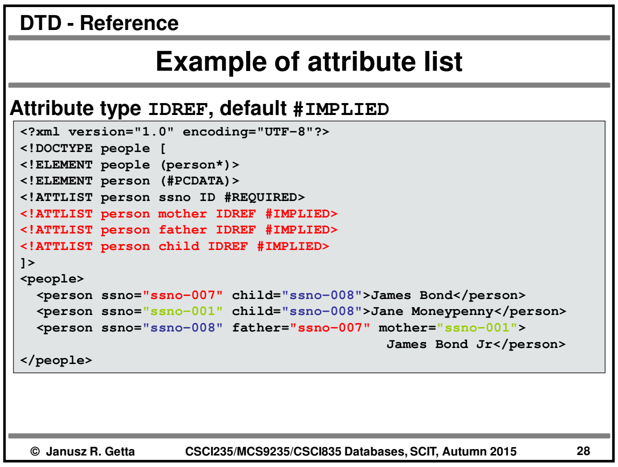
```
Ln 9 Col 22 - Attribute value "I" of type ID must be unique within the
document.
Ln 10 Col 11 - Attribute "number" is required and must be specified
for element type "chapter".
```



Example of attribute list

Attribute type `ID`, default `#REQUIRED`

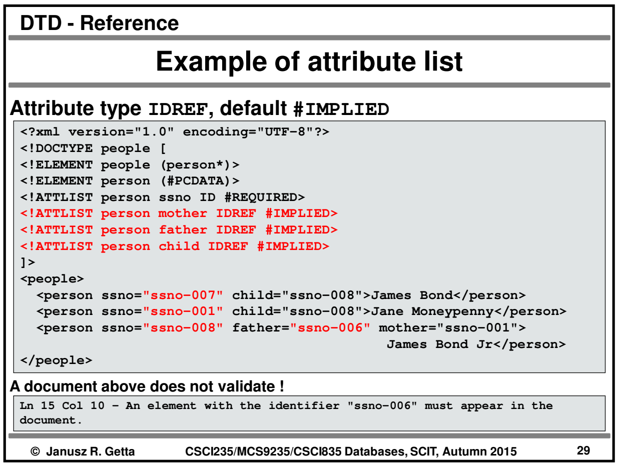
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
<!ELEMENT book (chapter+)>
<!ELEMENT chapter (#PCDATA)>
<!ATTLIST chapter number ID #REQUIRED>
]>
<book>
       <chapter number="I">Introduction</chapter>
       <chapter number="II">Data Modeling</chapter>
       <chapter number="III">Transaction processing</chapter>
</book>
```



Example of attribute list

Attribute type `IDREF`, default `#IMPLIED`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE people [
<!ELEMENT people (person*)>
<!ELEMENT person (#PCDATA)>
<!ATTLIST person ssno ID #REQUIRED>
<!ATTLIST person mother IDREF #IMPLIED>
<!ATTLIST person father IDREF #IMPLIED>
<!ATTLIST person child IDREF #IMPLIED>
]>
<people>
  <person ssno="ssno-007" child="ssno-008">James Bond</person>
  <person ssno="ssno-001" child="ssno-008">Jane Moneypenny</person>
  <person ssno="ssno-008" father="ssno-007" mother="ssno-001">
                                          James Bond Jr</person>
</people>
```



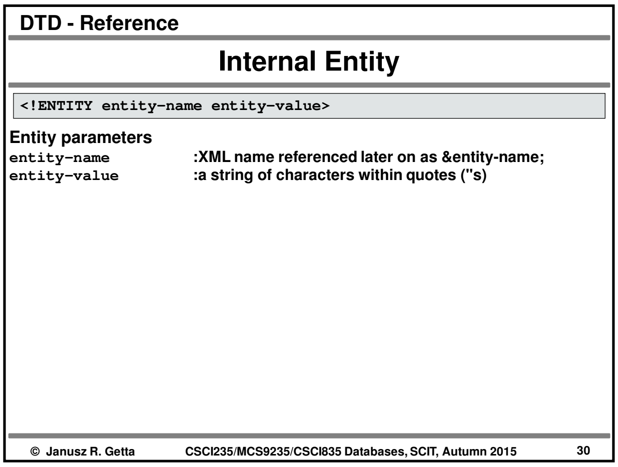
Example of attribute list

Attribute type `IDREF`, default `#IMPLIED`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE people [
<!ELEMENT people (person*)>
<!ELEMENT person (#PCDATA)>
<!ATTLIST person ssno ID #REQUIRED>
<!ATTLIST person mother IDREF #IMPLIED>
<!ATTLIST person father IDREF #IMPLIED>
<!ATTLIST person child IDREF #IMPLIED>
]>
<people>
  <person ssno="ssno-007" child="ssno-008">James Bond</person>
  <person ssno="ssno-001" child="ssno-008">Jane Moneypenny</person>
  <person ssno="ssno-008" father="ssno-006" mother="ssno-001">
                                          James Bond Jr</person>
</people>
```

A document above does not validate !

```
Ln 15 Col 10 - An element with the identifier "ssno-006" must appear in the
document.
```



Internal Entity

```
<!ENTITY entity-name entity-value>
```

Entity parameters

`entity-name` :XML name referenced later on as &entity-name;

`entity-value` :a string of characters within quotes ("s)

Example of Internal Entity

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE specification [
<!ELEMENT specification (#PCDATA)>
<!ENTITY prompt "SQL>">
<!ENTITY connect "CONNECT">
]>
<specification>
       Start SQLPlus client. The system should display prompt
       &prompt;. Type in: &connect; jrg/pswd. The system should
       reply with a prompt &prompt;. And so on, and so on ...
</specification>
```

External Entity

```
<!ENTITY entity-name SYSTEM url >
```

Entity parameters

`entity-name`	:XML name referenced later on as &entity-name;
`url`	:Universal Resource Locator where the externally parsed entity can be found

```
<!ENTITY entity-name PUBLIC public-id url>
```

Entity parameters

`entity-name`	:XML name referenced later on as &entity-name;
`public-id`	:public identifier, may be used to provide an alternate url
`url`	:Universal Resource Locator where the externally parsed entity can be found

Example of External Entity

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE warranty [
<!ELEMENT warranty (#PCDATA)>
<!ENTITY warranty-text SYSTEM "warranty-text.txt">
]>
<warranty>
       &warranty-text;
</warranty>
```

Example of External Entity

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE www-site [
<!ELEMENT www-site (html)>
<!ELEMENT html (#PCDATA)>
<!ENTITY html SYSTEM "http://www.uow.edu.au/~jrg/qqq.html">
]>
<www-site>
       &html;
</www-site>
```

`http://www.uow.edu.au/~jrg/qqq.html`

```
<html>
  Janusz Getta's Home Page
</html>
```

Example of External Entity

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE www-site [
<!ELEMENT www-site (html)>
<!ELEMENT html (#PCDATA)>
<!ENTITY html PUBLIC "http://www.uow.edu.au/~jrg/qqq.html"
                     "http://www.uow.edu.au/~jrg/index.html">
]>
<www-site>
       &html;
</www-site>
```

Parameter Entity

```
<!ENTITY % entity-name entity-value >
```

Entity parameters

`entity-name`	:XML name referenced later on as &entity-name;
`entity-value`	:a string of characters within quotes ("s)

Parameter entities can only be used in external DTDs !

Example of Parameter Entity

par-book.dtd

```
<!ENTITY % p "(#PCDATA)">
<!ELEMENT book (title, chapter+)>
<!ATTLIST book author CDATA #REQUIRED>
<!ELEMENT title %p;>
<!ELEMENT chapter %p;>
<!ATTLIST chapter id ID #REQUIRED>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM "par-book.dtd">
<book author="C.J.Date">
       <title>Databases</title>
       <chapter id="I">Introduction</chapter>
       <chapter id="II">Conceptual modeling</chapter>
       <chapter id="III">SQL</chapter>
</book>
```

Namespaces in DTDs

```
<sql:select-stmt xmlns:sql="http://www.sql.org/">
       <sql:select dialect="SQL Server">*</sql:select>
       <sql:from>EMPLOYEE</sql:from>
</sql:select-stmt>
```

```
<!DOCTYPE sql:select-stmt [
<!ELEMENT sql:select-stmt (sql:select,sql:from,sql:where?)>
<!ATTLIST sql:select-stmt xmlns:sql CDATA #FIXED
                                        "http://www.sql.org/">
<!ELEMENT sql:select (#PCDATA)>
<!ATTLIST sql:select sql:dialect CDATA #REQUIRED>
<!ELEMENT sql:from (#PCDATA)>
<!ELEMENT sql:where (#PCDATA)>
]>
<sql:select-stmt xmlns:sql="http://www.sql.org/">
       <sql:select dialect="SQL Server">*</sql:select>
       <sql:from>EMPLOYEE</sql:from>
</sql:select-stmt>
```

Limitations of DTD

(1) It is not itself using XML syntax (the SGML heritage can be very unintuitive + if using XML, DTDs could potentially themselves be syntax checked with a "meta DTD")

(2) Mixed into the XML 1.0 spec (would be much less confusing if specified separately + even non-validating processors must look at the DTD)

(3) No constraints on character data (if character data is allowed, any character data is allowed)

…

Limitations of DTD

…

(4) Too simple attribute value models (enumerations are clearly insufficient)

(5) Cannot mix character data and regexp content models (and the content models are generally hard to use for complex requirements)

(6) No support for Namespaces (of course, XML 1.0 was defined before Namespaces)

(7) Very limited support for modularity and reuse (the entity mechanism is too low-level)

…

Limitations of DTD

…

(8) No support for schema evolution, extension, or inheritance of declarations (difficult to write, maintain, and read large DTDs, and to define families of related schemas)

(9) Limited white-space control (xml:space is rarely used)

(10) No embedded, structured self-documentation (<!-- comments --> are not enough)

(11) Content and attribute declarations cannot depend on attributes or element context (many XML languages use that, but their DTDs have to "allow too much")

…

Limitations of DTD

…

(12) Too simple ID attribute mechanism (no points-to requirements, uniqueness scope, etc.)

(13) Only defaults for attributes, not for elements (but that would often be convenient)

(14) Cannot specify "any element" or "any attribute" (useful for partial specifications and during schema development)

(15) Defaults cannot be specified separate from the declarations (would be convenient to have defaults in separate modules)

Limitations of DTD

So, if DTD is so imperfect then what other choice do we have ?

Appendix: regular expressions

An alphabet is a finite set of symbols

For example, a set `{0,1}` is an alphabet; it consists of two symbols: `0` and `1`

A string is a finite sequence of symbols taken from agiven alphabet

For example, `01110011` is a string built from the symbols in alphabet `{0,1}`

A language is a set of all string that can be formed from the symbols of a given alphabet using a given collection of rules

Regular expressions is a formalism that can be used to define certain classes of languages

Appendix: regular expressions

A regular expression is an expression constructed accordingly to the following rules:

(1) An empty symbol ε is a regular expression

(2) For each symbol `a` that belongs to an alphabet `a` is a regular expression

(3) If `x` is a regular expression and `y` is a regular expression than `xy` (concatenation) is a regular expression

(4) If `x` is a regular expression and `y` is a regular expression than `x|y` (alternation) is a regular expression

(5) If `x` is a regular expression then `x*` is a regular expression

(6) If `x` is a regular expression then `x+` is a regular expression

(7) If `x` is a regular expression then `(x)` is a regular expression

Appendix: regular expressions

The semantics of regular expressions are as follows:

(1) An empty symbol ε defines an empty language

(2) Symbol `a` defines a language that consist of one string `a`

(3) If `x` defines a language L_x and `y` defines a language L_y then `xy` (concatenation) defines a language that consists of all strings obtained from the concatenation of strings in L_x with the strings in L_y

(4) If `x` defines a language L_x and `y` defines a language L_y than than `x|y` (alternation) defines a language obtained from the union L_x and L_y

(5) `x* = ε | x | xx | xxx | …`

(6) `x+ = x | xx | xxx | …`

Appendix: regular expressions

A regular expression ε defines a language `L = {ε}`

A regular expression `abc` defines a language `L = {abc}`

A regular expression `abc|cde` defines a language
`L = {abc, cde}`

A regular expression `a(bc|de)` defines a language
`L = {abc, ade}`

A regular expression `ab*` defines a language
`L = {a, ab, abb, abbb, abbbb, …}`

A regular expression `(a|b)*` defines a language
`L = {ε, a, b, aa, ab, ba, bb, aaa, aba, baa, bba, aab, abb, bab, bbb, … }`

A regular expression `letter|(letter|digit)*` defines a set of strings that start from a `letter` and consist of `letter`s and/or `digits`