Applied Databases

Lecture 4
SAX Parsing, Entity Relationship Model

Sebastian Maneth

University of Edinburgh  -  January 26\textsuperscript{st}, 2017
Outline

1. **SAX**  Simple API for XML
2. Comments wrt  **Assignment 1**
3. Data Redundancy Problem
4. Entity Relationship Model
1. SAX – Simple API for XML

Recall one of the promises of XML:

→ you never need to write a parser again.

→ if you want to build up your own (e.g. memory-efficient) data structure, you need to “talk” to the parser.

The XML parser should give you low level access to the data:

→ tag by tag

→ text-node by text-node.

in “document order”.
1. SAX – Simple API for XML

Recall one of the promises of XML:

→ you never need to write a parser again.

→ if you want to build up your own (e.g. memory-efficient) data structure, you need to “talk” to the parser.

The XML parser should give you low level access to the data:

→ tag by tag

→ text-node by text-node.

in “document order”.  ➔ SAX
SAX—Simple API for XML

- **SAX** (Simple API for XML) is, unlike DOM, *not* a W3C standard, but has been developed jointly by members of the XML-DEV mailing list (*ca.* 1998).
- SAX processors use **constant space**, regardless of the XML input document size.
  - Communication between the SAX processor and the backend XML application does *not* involve an intermediate tree data structure.
  - Instead, the **SAX parser sends events** to the application whenever a certain piece of XML text has been recognized (*i.e.*, parsed).
  - The **backend acts on/ignores events** by populating a **callback function table**.

---

7http://www.saxproject.org/
SAX—Simple API for XML

- **SAX**\(^7\) (*Simple API for XML*) is, unlike DOM, *not* a W3C standard, but has been developed jointly by members of the XML-DEV mailing list (*ca.* 1998).
- SAX processors use **constant space**, regardless of the XML input document size.
  - Communication between the SAX processor and the backend XML application does *not* involve an intermediate tree data structure.
  - Instead, the **SAX parser sends events** to the application whenever a certain piece of XML text has been recognized (*i.e.*, parsed).
  - The **backend acts on/ignores events** by populating a **callback function table**.

---

\(^7\)http://www.saxproject.org/
Sketch of SAX’s mode of operations

- A SAX processor reads its input document **sequentially** and once only. (except open tags & DTD-relevant data)
- No memory of what the parser has seen so far is retained while parsing. As soon as a **significant bit of XML text** has been recognized, an **event** is sent.
- The application is able to act on events **in parallel** with the parsing progress.
SAX Events

To meet the constant memory space requirement, SAX reports **fine-grained parsing events** for a document:

<table>
<thead>
<tr>
<th>Event</th>
<th>...reported when seen</th>
<th>Parameters sent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>startDocument</code></td>
<td><code>&lt;?xml...?&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>endDocument</code></td>
<td><code>EOF</code></td>
<td></td>
</tr>
<tr>
<td><code>startElement</code></td>
<td><code>&lt;t a_1=v_1 ... a_n=v_n&gt;</code></td>
<td><code>t, (a_1, v_1), ..., (a_n, v_n)</code></td>
</tr>
<tr>
<td><code>endElement</code></td>
<td><code>&lt;/t&gt;</code></td>
<td><code>t</code></td>
</tr>
<tr>
<td><code>characters</code></td>
<td><code>text content</code></td>
<td><code>Unicode buffer ptr, length</code></td>
</tr>
<tr>
<td><code>comment</code></td>
<td><code>&lt;!--c--&gt;</code></td>
<td><code>c</code></td>
</tr>
<tr>
<td><code>processingInstruction</code></td>
<td><code>&lt;?t pi?&gt;</code></td>
<td><code>t, pi</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**N.B.**: Event `startDocument` is sent even if the optional XML text declaration should be missing.
SAX Events

To meet the constant memory space requirement, SAX reports fine-grained parsing events for a document:

<table>
<thead>
<tr>
<th>Event</th>
<th>...reported when seen</th>
<th>Parameters sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>startDocument</td>
<td>&lt;?xml...?&gt;(^8)</td>
<td></td>
</tr>
<tr>
<td>endDocument</td>
<td>⟨EOF⟩</td>
<td></td>
</tr>
<tr>
<td>startElement</td>
<td>⟨t a₁=v₁ ... aₙ=vₙ⟩</td>
<td>t, (a₁, v₁), ..., (aₙ, vₙ)</td>
</tr>
<tr>
<td>endElement</td>
<td>⟨/t⟩</td>
<td>t</td>
</tr>
<tr>
<td>characters</td>
<td>text content</td>
<td>Unicode buffer ptr, length</td>
</tr>
<tr>
<td>comment</td>
<td>&lt;!--c--&gt;</td>
<td>c</td>
</tr>
<tr>
<td>processingInstruction</td>
<td>⟨?t pi?⟩</td>
<td>t, pi</td>
</tr>
</tbody>
</table>

\(^8\)N.B.: Event startDocument is sent even if the optional XML text declaration should be missing.

Cave: Characters of one text node may be sent in several chunks
dilbert.xml

```xml
<?xml encoding="utf-8"?> *
<bubbles> *2
    <!-- Dilbert looks stunned --> *3
    <bubble speaker="phb" to="dilbert"> *4
        Tell the truth, but do it in your usual engineering way
        so that no one understands you. *5
    </bubble> *6
</bubbles> *7 *8
```

<table>
<thead>
<tr>
<th>Event</th>
<th>Parameters sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>*1</td>
<td>startDocument</td>
</tr>
<tr>
<td>*2</td>
<td>startElement</td>
</tr>
<tr>
<td>*3</td>
<td>comment</td>
</tr>
<tr>
<td>*4</td>
<td>startElement</td>
</tr>
<tr>
<td>*5</td>
<td>characters</td>
</tr>
<tr>
<td>*6</td>
<td>endElement</td>
</tr>
<tr>
<td>*7</td>
<td>endElement</td>
</tr>
<tr>
<td>*8</td>
<td>endDocument</td>
</tr>
</tbody>
</table>

9 Events are reported in **document reading order** *1, *2, ..., *8.

10 **N.B.**: Some events suppressed (white space).
SAX and the XML Tree Structure

- Looking closer, the **order** of SAX events reported for a document is determined by a **preorder traversal** of its document tree:\(^{12}\):

```
1
2 <a>2
3   <b>3 foo4 </b>5
4   </a>7
5 <c>7
6   <d>8 bar9 </d>10
7       <e>11 baz12 </e>13
8     </c>14
9   </a>15 16
```

**N.B.:** An *Elem* [*Doc*] node is associated with two SAX events, namely *startElement* and *endElement* [*startDocument*, *endDocument*].

---

\(^{12}\)Sequences of sibling *Char* nodes have been collapsed into a single *Text* node.
public void startElement(String nameSpaceURI, String localName, String rawName, Attributes atts) throws SAXException {
    System.out.println("Opening tag: " + localName);
    // Show attributes, if any
    if (atts.getLength() > 0)
        for (int index = 0; index < atts.getLength(); index++)
            System.out.println("Attribute: "+ atts.getLocalName(index) + " = " + atts.getValue(index));
}

public void endElement(String nameSpaceURI, String localName, String rawName) throws SAXException {
    System.out.print("Closing tag : " + localName);
    System.out.println();
}

// Character data handling
public void characters(char[] ch, int start, int end) throws SAXException {
    System.out.println("#PCDATA: " + new String(ch, start, end));
}
<?xml version="1.0"?>
<!DOCTYPE greeting [
<!ENTITY hi “Hello”>
<!ENTITY hi1 “&hi;&hi;”>
<!ENTITY hi2 “&hi1;&hi1;”>
<!ENTITY hi3 “&hi2;&hi2;”>
<!ENTITY s “<d></d>”>
]>
<a a1='17' a2='29'>
<b>xy &hi3; world &s; zz</b></a>

// Show attributes, if any
if (atts.getLength() > 0) {
    for (int index = 0; index < atts.getLength(); index++)
        System.out.println("Attribute: atts.getLocalName(index)
            + ";" + atts.getValue(index));
}

$ java MySAXApp file.xml
Start document
Start element: a
Attribute: a1=17
Attribute: a2=29
Start element: b
Characters: "xy "
Characters: "Hello"
Characters: "Hello"
Characters: "Hello"
Characters: "Hello"
Characters: "Hello"
Characters: "Hello"
Characters: "Hello world "
Start element: d
End element: d
Characters: " zz"
End element: b
End element: a
End document
$ java MySAXApp file.xml
Start document
Start element: a
Attribute: a1=17
Attribute: a2=29
Start element: b
Characters: "xy "
Characters: "Hello"
Characters: "Hello"
Characters: "Hello"
Characters: "Hello"
Characters: "Hello"
Characters: "Hello"
Characters: "Hello world "
Start element: d
End element: d
Characters: " zz"
End element: b
End element: a
End document
2. Comments for Assignment 1
2. Comments for Assignment 1

→ You do not need the VirtualBox image that we provide to implement your converter!

→ You only need a Java JDK (javac and java) and SAX or DOM (org.xml.sax.* and org.w3c.dom.*)

→ Only when you start to import into MySQL, it might be convenient to use the image, because it has a MySQL server running for you.
2. Comments for Assignment 1

→ Download and install VirtualBox

→ Download adAssignment1_Ubuntu32.vdi.zip from assignment web page

→ Unzip this file (this may take a while!) to obtain the VirtualBox disk image adAssignment1_Ubuntu32.vdi

→ Run VirtualBox. Click Machine → New
  - give your new machine a name,
  - select Type “Linux” and Version “Ubuntu (32-bit)”
  - then select a Memory size (e.g., 512MB or 768MB)
  - then click “Use an existing virtual hard drive file”
  - click on the folder icon and select your adAssignment1_Ubuntu32.vdi

→ Now click on the machine, on top left, and then click “Start” from top
  The image will now boot, this may take a while.
→ Press **CTRL-ALT-T** to open a **terminal**
(double click on top-bar to make terminal full-screen)

→ we assume some rudimentary knowledge of Unix shell commands
  (e.g. `ls`, `cd`, `less`, `vi`)

(use **CTRL-+** and **–** to increase/decrease font size)

If you like to use another editor, e.g., emacs, then install it via:

→ **sudo apt-get install emacs**

(no password required)
```
ad@ad-VirtualBox:~$ ls -l
 total 20
 drwxr-xr-x 2 ad  ad  4096 Jan 17 17:27 AD_Assignment_1
 drwxrwxr-x 2 ad  ad  4096 Jan 15 16:00 AD_Assignment_2
 drwxr-xr-x 2 ad  ad  4096 Jan  5 18:48 Desktop
 drwxrwxr-x 2 ad  ad  4096 Jan 17 17:27 ebay_data
 drwxrwxr-x 3 ad  ad  4096 Jan 15 16:03 svn

ad@ad-VirtualBox:~$ wget http://www.inf.ed.ac.uk/teaching/courses/ad/assignment_1/ebay-data.zip
--2016-01-17 17:29:18--  http://www.inf.ed.ac.uk/teaching/courses/ad/assignment_1/ebay-data.zip
Resolving www.inf.ed.ac.uk (www.inf.ed.ac.uk)... 129.215.33.176
Connecting to www.inf.ed.ac.uk (www.inf.ed.ac.uk)|129.215.33.176|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10414553 (9.9M) [application/zip]
Saving to: 'ebay-data.zip'

100%[================================================================================================] 10,414,553 53.0MB/s in 0.2s

2016-01-17 17:29:19 (53.0 MB/s) - 'ebay-data.zip' saved [10414553/10414553]

ad@ad-VirtualBox:~$ unzip -d SEBAY_DATA ebay-data.zip
Archive: ebay-data.zip
  inflating: /home/ad/ebay_data/items-0.xml
  inflating: /home/ad/ebay_data/items-10.xml
  inflating: /home/ad/ebay_data/items-11.xml
  inflating: /home/ad/ebay_data/items-12.xml
  inflating: /home/ad/ebay_data/items-13.xml
  inflating: /home/ad/ebay_data/items-14.xml
  inflating: /home/ad/ebay_data/items-15.xml
  inflating: /home/ad/ebay_data/items-16.xml
  inflating: /home/ad/ebay_data/items-17.xml
  inflating: /home/ad/ebay_data/items-18.xml
  inflating: /home/ad/ebay_data/items-19.xml
  inflating: /home/ad/ebay_data/items-20.xml
  inflating: /home/ad/ebay_data/items-21.xml
  inflating: /home/ad/ebay_data/items-22.xml
  inflating: /home/ad/ebay_data/items-23.xml
  inflating: /home/ad/ebay_data/items-24.xml
  inflating: /home/ad/ebay_data/items-25.xml
```
ad@ad-VirtualBox:~$ cd AD_Assignment_1
ad@ad-VirtualBox:~/AD_Assignment_1$ ls
ad@ad-VirtualBox:~/AD_Assignment_1$ wget http://www.inf.ed.ac.uk/teaching/courses/ad --2016-01-17 17:33:01-- http://www.inf.ed.ac.uk/teaching/courses/ad/assignment_1/MyResolving www.inf.ed.ac.uk (www.inf.ed.ac.uk)... 129.215.33.176
Connecting to www.inf.ed.ac.uk (www.inf.ed.ac.uk)|129.215.33.176|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5588 (5.5K) [text/plain]
Saving to: ‘MyDOM.java’

100%

2016-01-17 17:33:01 (438 MB/s) - ‘MyDOM.java’ saved [5588/5588]

ad@ad-VirtualBox:~/AD_Assignment_1$ javac MyDOM.java
ad@ad-VirtualBox:~/AD_Assignment_1$ ls
MyDOM.class MyDOM.java MyDOM$MyErrorHandler.class
ad@ad-VirtualBox:~/AD_Assignment_1$ java MyDOM $EBAY_DATA/items-0.xml | less
ad@ad-VirtualBox:~/AD_Assignment_1$
Successfully parsed /home/ad/ebay_data/items-0.xml
Type = Document, Name = #document, Value = null
  Type = Element, Name = Items, Value = null
    Type = Text, Name = #text, Value =
      Type = Element, Name = Item, Value = null
        Type = Attr, Name = ItemID, Value = 1043374545
          Type = Text, Name = #text, Value = 1043374545
        Type = Text, Name = #text, Value =
      Type = Element, Name = Name, Value = null
        Type = Text, Name = #text, Value = christopher radko | fritz n_ frosty sledding
      Type = Text, Name = #text, Value =
    Type = Element, Name = Category, Value = null
      Type = Text, Name = #text, Value = Collectibles
    Type = Text, Name = #text, Value =
  Type = Element, Name = Category, Value = null
    Type = Text, Name = #text, Value = Decorative & Holiday
  Type = Text, Name = #text, Value =
  Type = Element, Name = Category, Value = null
    Type = Text, Name = #text, Value = Decorative by Brand
  Type = Text, Name = #text, Value =
  Type = Element, Name = Category, Value = null
    Type = Text, Name = #text, Value = Christopher Radko
  Type = Text, Name = #text, Value =
  Type = Element, Name = Currently, Value = null
    Type = Text, Name = #text, Value = $30.00
wget http://www.inf.ed.ac.uk/teaching/courses/ad/java
--2016-01-17 17:39:08--  http://www.inf.ed.ac.uk/teaching/courses/ad/assignment_1/My
Resolving www.inf.ed.ac.uk (www.inf.ed.ac.uk)... 129.215.33.176
Connecting to www.inf.ed.ac.uk (www.inf.ed.ac.uk)|129.215.33.176|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3263 (3.2K) [text/plain]
Saving to: ‘MySAX.java.1’

100%[=============================================] 3,263  -

2016-01-17 17:39:08 (228 MB/s) - ‘MySAX.java.1’ saved [3263/3263]

javac MySAX.java
java MySAX $EBAY_DATA/items-0.xml | less
Start document
Start element: Items
Characters: "\n 
Start element: Item
Attribute: ItemID=1043374545
Characters: "\n 
Start element: Name
Characters: "christopher radko | fritz n_ frosty sledding"
End element: Name
Characters: "\n 
Start element: Category
Characters: "Collectibles"
End element: Category
Characters: "\n 
Start element: Category
Characters: "Decorative"
Characters: 
&
Characters: " Holiday"
End element: Category
Characters: "\n 
Start element: Category
Characters: "Decorative by Brand"
End element: Category
Characters: "\n 
Start element: Category
Characters: "Christopher Radko"
The contents of items-0.xml:

```xml
<Items>
  <Item ItemID="1043374545">
    <Name>christopher radko | fritz n_ frosty sledding</Name>
    <Category>Collectibles</Category>
    <Category>Decorative &amp; Holiday</Category>
  </Item>
  ...
</Items>
```
3. Data Redundancy Problem

```xml
<!ELEMENT Items            (Item*)>
<!ELEMENT Item             (Name, Category+, Currently, Buy_Price?,
                            First_Bid, Number_of_Bids,
                            Bids, Location, Country, Started, Ends,
                            Seller, Description)>
<!ELEMENT Bids             (Bid*)>
    <!ELEMENT Bid          (Bidder, Time, Amount)>
    <!ATTLIST Bidder
                     UserID CDATA #REQUIRED
                     Rating CDATA #REQUIRED>
<!ELEMENT Seller           EMPTY>
    <!ATTLIST Seller
                     UserID CDATA #REQUIRED
                     Rating CDATA #REQUIRED>
```

![Diagram](image)

UserID="!peanut"
Rating=58823

UserID="!peanut"
Rating=58823
From *items.txt*:

Bids/Bid/Bidder: Attribute UserID uniquely identifies a user. Attribute Rating is the bidder's rating. Location and Country child elements have the bidder's location information. 

Note that a user may bid on multiple items. In that case, his Rating, Location, and Country information are the same across all bids.

[...]

Seller: Attributes give the seller's UserID and Rating. A user may have two separate ratings as a bidder and a seller, in case the seller is also bidding on other item(s).
<!ELEMENT Bids (Bid*)>
<!ELEMENT Bid (Bidder, Time, Amount)>
<!ATTLIST Bidder UserID CDATA #REQUIRED
Rating CDATA #REQUIRED>
<!ELEMENT Seller EMPTY>
<!ATTLIST Seller UserID CDATA #REQUIRED
Rating CDATA #REQUIRED>
If you store the data in XML (as given) what are possible problems?
→ If you store the data in XML (as given) what are possible problems?
→ Why is **data redundancy** a problem?
→ Why is data redundancy a problem?

→ Imagine later do want to change a Rating (say, in a DB of only open auctions)
→ Updating redundant copies:

→ All Sellers/Bidders to be updated have to be locked and updated “at once” to guarantee *consistency*

→ Expensive!! (generates “out-time”)

Redundancy Problem!
→ Data redundancy leads to data anomalies and corruption.

→ Data redundancy should be avoided by design!

→ In our XML example, how could the Rating-redundancy be removed?
→ Data redundancy leads to data anomalies and corruption.

→ Data redundancy should be avoided by design!

→ in our XML example, how could the Rating-redundancy be removed?

Ratings appear only once!

```xml
<!ATTLIST User UserID ID #REQUIRED>
<!ATTLIST User Rating CDATA #REQUIRED>
<!ATTLIST Bidder BidderID IDREF #REQUIRED>
<!ATTLIST Seller SellerID IDREF #REQUIRED>
```
Issue with the ID/IDREF solution:

→ Where are UserID-entries kept in the tree? (arbitrary / ‘tree-implementation-detail’)

→ ID-attribute must contain an XML name that is unique within the document; more precisely: no other ID-attribute in the document may have the same value.

Ratings appear only once!

```xml
<!ATTLIST User UserID ID #REQUIRED>
<!ATTLIST User Rating CDATA #REQUIRED>
<!ATTLIST Bidder BidderID IDREF #REQUIRED>
<!ATTLIST Seller SellerID IDREF #REQUIRED>
```
Issue with the ID/IDREF solution:

→ Where are UserID-entries kept in the tree? (arbitrary / ‘tree-implementation-detail’)

→ ID-attribute must contain an XML name that is unique within the document; more precisely: no other ID-attribute in the document can have the same value.

→ thus, if ItemID was an ID-attribute, then each entry would have to be different from any UserID!
→ Why? Why would it be satisfied in the data?

UserID="nobody138"
Rating=427

BidderID="nobody138"

<!ATTLIST User UserID ID #REQUIRED>
<!ATTLIST User Rating CDATA #REQUIRED>
<!ATTLIST Bidder BidderID IDREF #REQUIRED>
<!ATTLIST Seller SellerID IDREF #REQUIRED>

Ratings appear only once!
Issue with the ID/IDREF solution:

- Where are UserID-entries kept in the tree? (arbitrary / ‘tree-implementation-detail’)

- ID-attribute must contain an XML name that is unique within the document; more precisely: no other ID-attribute in the document can have the same value.

- thus, if ItemID was an ID-attribute, then each entry would have to be different from any UserID!
  - Why? Why would it be satisfied in the data?

- On EBAY data this solution does NOT work! (because of XML name issues)
<!DOCTYPE greeting [
   <!ELEMENT greeting (user | bidder | seller)>]
<!ELEMENT user EMPTY>
<!ATTLIST user BidderID ID #REQUIRED>
<!ATTLIST user Rating CDATA #REQUIRED>
<!ELEMENT bidder EMPTY>
<!ATTLIST bidder BidderID IDREF #REQUIRED>
]
<greeting>
   <user BidderID="!peanut" rating="427"/>
   <seller BidderID="!peanut"/>
</greeting>

test.xml

$ xml-xparse -n test.xml
Attempting validating, namespace-ignorant parse
Error: file:/home/ad/test.xml:11:48: Attribute value "!peanut" of type ID must be a name.
Error: file:/home/ad/test.xml:11:76: Attribute value "!peanut" of type IDREF must be a name.
Parse succeeded (0.37) with 2 errors and no warnings.
$
[1]  \textbf{document} ::= prolog element Misc*
[2]  Char ::= a Unicode character
[3]  S ::= (\textquoteleft \textquoteleft | \textquoteleft \textbackslash t\textquoteleft | \textquoteleft \textbackslash n\textquoteleft | \textquoteleft \textbackslash r\textquoteleft\textquoteleft)+
[4]  NameChar ::= (Letter | Digit | \textquoteleft .\textquoteleft | \textquoteleft -\textquoteleft | \textquoteleft :\textquoteleft)
[5]  Name ::= (Letter | \textquoteleft _\textquoteleft | \textquoteleft :\textquoteleft) (NameChar)*
[84]  Letter ::= [a-zA-Z]
[88]  Digit ::= [0-9]

→ Name must start with a-zA-Z or with \_ or with \:
(also: no white space!)

→ BidderID may not equal !peanut

→ in presence of namespaces, must even be an \textbf{NCName}

\textbf{NCName} ::= Name - (Char* \textquoteleft :\textquoteleft Char*)
Issue with the ID/IDREF solution:

→ On the EBAY-data, solution does not work (because of XML names)!

→ Would need to introduce additional IDs that are allowed
  (→ one more level of indirection)

```xml
<!DOCTYPE greeting [ 
  <!ELEMENT greeting (user | bidder | seller)*> 
  <!ELEMENT user EMPTY> 
  <!ATTLIST user Bidder_ID ID #REQUIRED> 
  <!ATTLIST user BidderID CDATA #REQUIRED> 
  <!ATTLIST user Rating CDATA #REQUIRED> 
  <!ELEMENT bidder EMPTY> 
  <!ATTLIST bidder BidderID IDREF #REQUIRED> 
]> 
<greeting> 
  <user Bidder_ID="u127" BidderID="!peanut" Rating="427"/> 
  <bidder Bidder_ID="u127"/> 
</greeting>
```

unique wrt all ID-attribute values!
Issue with the ID/IDREF solution:

→ On the EBAY-data, solution does not work (because of XML names)!

→ Would need to introduce additional IDs that are allowed
  (→ one more level of indirection)

→ Similar to an ‘implementation’ of a table of this form:

<table>
<thead>
<tr>
<th>.</th>
<th>.</th>
<th>.</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>u127</td>
<td>!peanut</td>
<td>427</td>
<td></td>
</tr>
</tbody>
</table>

→ In a table (of a database), u127 can simply be 127
Proposed Solution

→ use XML to exchange data, not to store or query it
→ store data in tables of a database
→ query the tables using SQL
Questions

→ introduce new integer-ID column: yes or no?

→ how to declare that a column is of type ID?

→ does every table have an ID column?

→ can there be duplicates of tuples (rows) in a table?

→ how can we check if our tables contain redundancy?

→ how can we express additional constraints that hold on the data? (e.g., end-time is after start-time)
Roadmap

→ Entity-Relationship Model (short)
  - define primary key ("ID column") in an abstract setting

→ Define data redundancy

→ Define functional dependencies

→ Define normal forms
4. Entity Relationship Model

→ high-level database model [Peter Chen (MIT) TODS 1, 1976]

→ useful for design before moving to a lower level model (e.g. relational)

ER Model has

→ Structural part
  - entity types
  - attributes
  - relationship types

→ Integrity constraints
  - primary keys for entity and relationship types
  - multiplicity constraints for relationship types

Next slides from Peter Wood’s DB Management Lecture
4. Entity Relationship Model

→ high-level database model  

[Peter Chen (MIT) TODS 1, 1976]

→ useful for design before moving to a lower level model (e.g. relational)

ER Model has

→ Structural part
  - entity types
  - attributes
  - relationship types

→ Integrity constraints
  - primary keys for entity and relationship types
  - multiplicity constraints for relationship types

ER Diagrams

→ relatively simple
→ user-friendly
→ unified view of data, independent of any implemented data mode.
Entity Types

**Entity** = a “thing” that exists and can be uniquely identified, e.g. an individual person

**Entity type** = collection of similar entities, e.g., a collection of people (rectangle)

Entity type has attributes (circles), representing properties of the entities.

Each Person has single Name, Address, and Nat. Insurance number (NI#)  
Each Person can have many Phones
Relationship Types

Relationship Type = association between two or more entity types. (diamond)

Multiplicity Constraints in Relationship Types

→ Many-to-One (or One-to-Many)
   An Employee Works in one Department or a Department has many Employees.

→ One-to-One
   A Manager Heads one Department and vice versa.

→ Many-to-Many
   A Lecturer Teaches many Students and a Student is Taught by many Lecturers
Example of Many-to-One Relationship Type

The arrowhead is drawn at the “one” end of rel. type

→ Each Employee Works-in one Department

→ Each Department has many Employees Working in it.
Example of One-to-One Relationship Type

The arrowhead is drawn at both ends

→ Each Manager Occupies one Office

→ Each Office has one Manager Occupying it
Example of Many-to-Many Relationship Type

No arrowheads

→ Each Lecturer Teaches many Students

→ Each Student is taught by many Lecturers
Multiple Relationship Types

Diagram:
- Diamond labeled "Teaches" connected to:
  - Rectangle labeled "Lecturer"
  - Diamond labeled "Tutors"
- Rectangle labeled "Student"
Participation Constraints in Relationships

→ optional (our default, sometimes indicated by multiplicity constraint 0..*)
e.g. Employee may or may not be assigned to a Department

→ mandatory (double lines, or multiplicity constraint 1..*)

→ some Lecturers may not Teach any Students
→ each Student *must* be taught by at least one Lecturer
Multiway Relationship Types

→ each supplier may supply different parts to different projects
END
Lecture 4