



| IBM Software Group

IMS19

## IMS v9 XML-DB

Christopher Holtz

IMS Development

Silicon Valley Laboratory

IBM Corporation



ON DEMAND BUSINESS™

©2005 IBM Corporation

## Overview

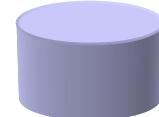
- Introduction
  - What is IMS XML-DB?
  - What is XML?
  - What are XML Schemas?
- IMS XML-DB Methodology (XML Visualization)
  - Decomposed Storage
  - Intact Storage
- IMS Java / XML-DB Tooling
  - Metadata Generation
- IMS XML-DB Java Implementation
  - SQL UDF Interface
  - Future Direction



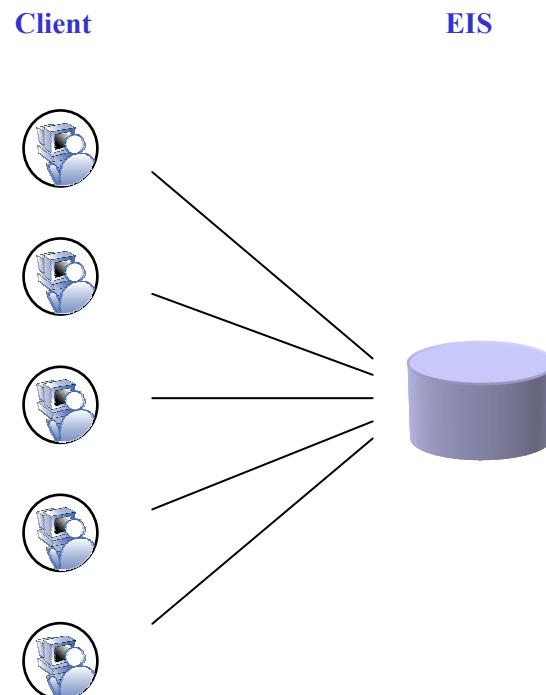
# Client Server Architecture

**Client**

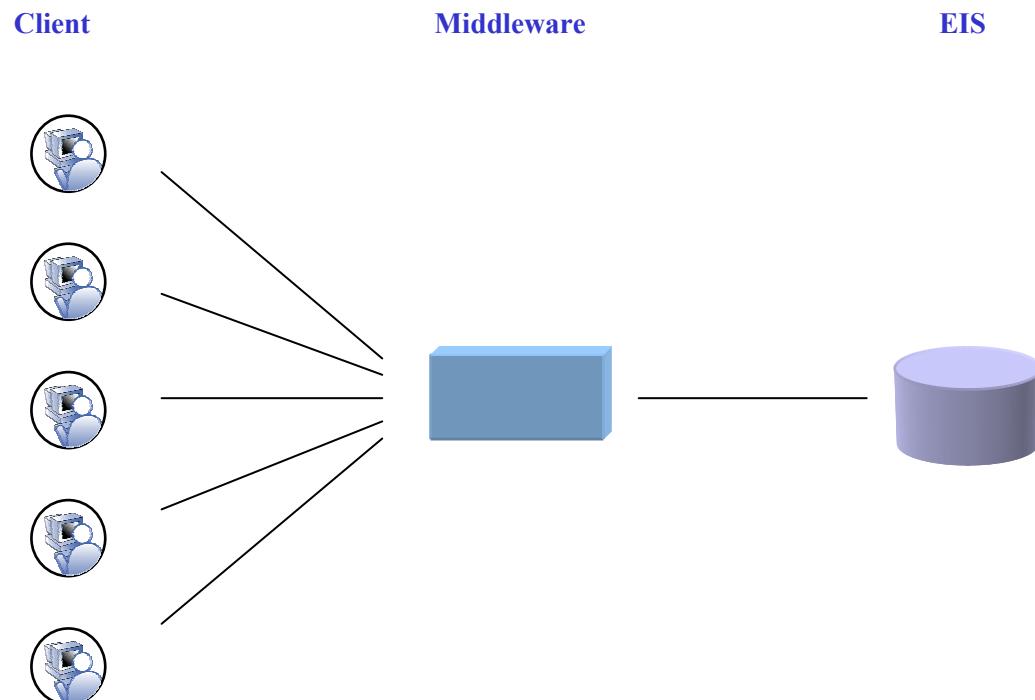
**EIS**



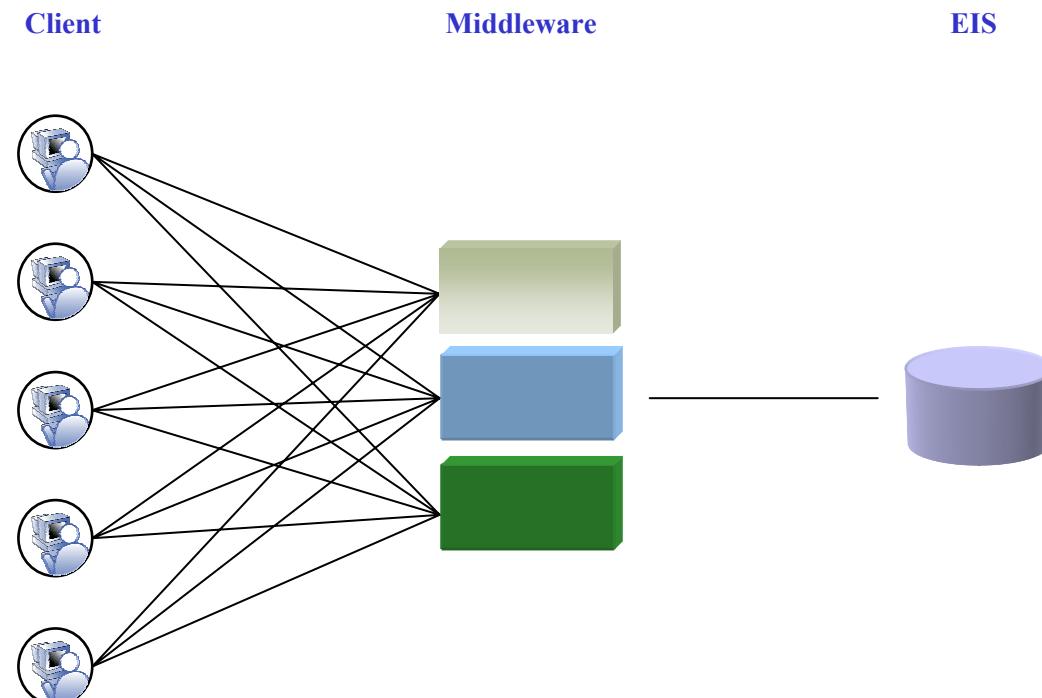
## Client Server Architecture



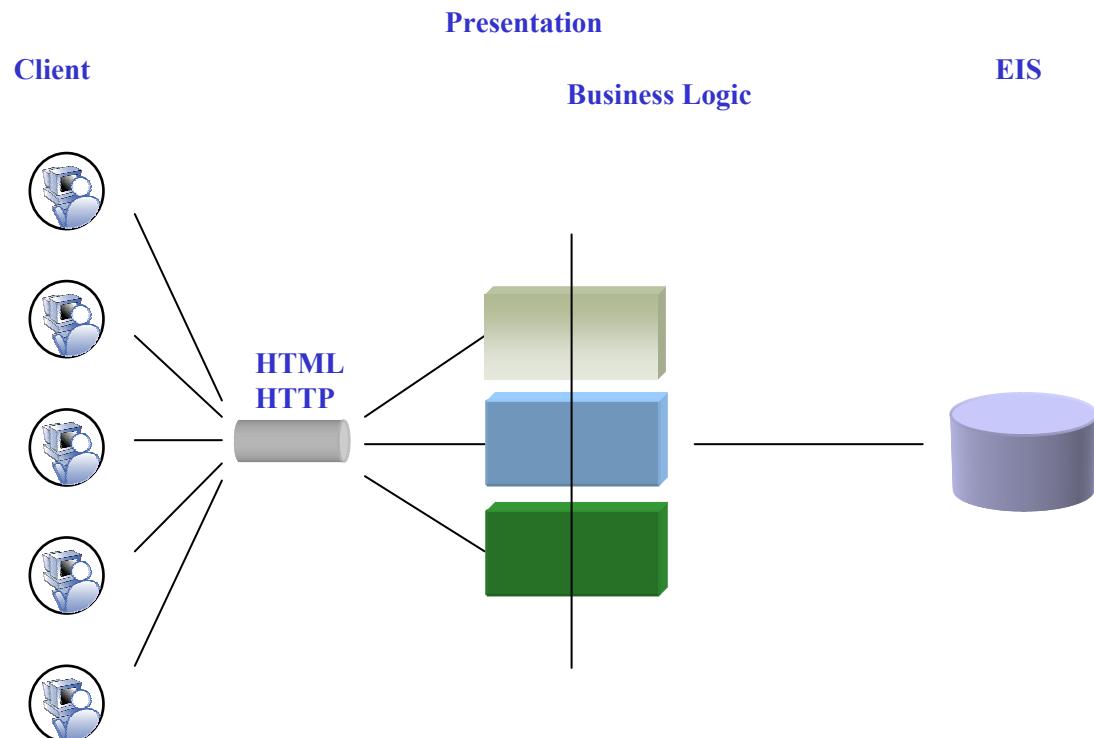
## 3-tier Architecture



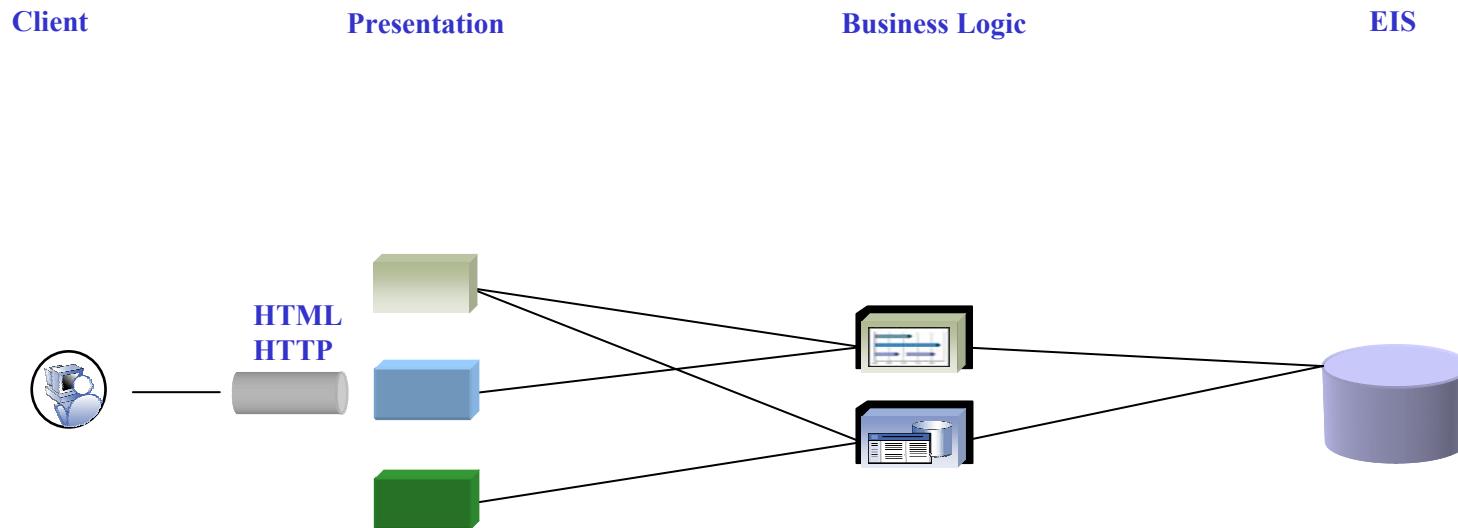
## 3-tier Architecture



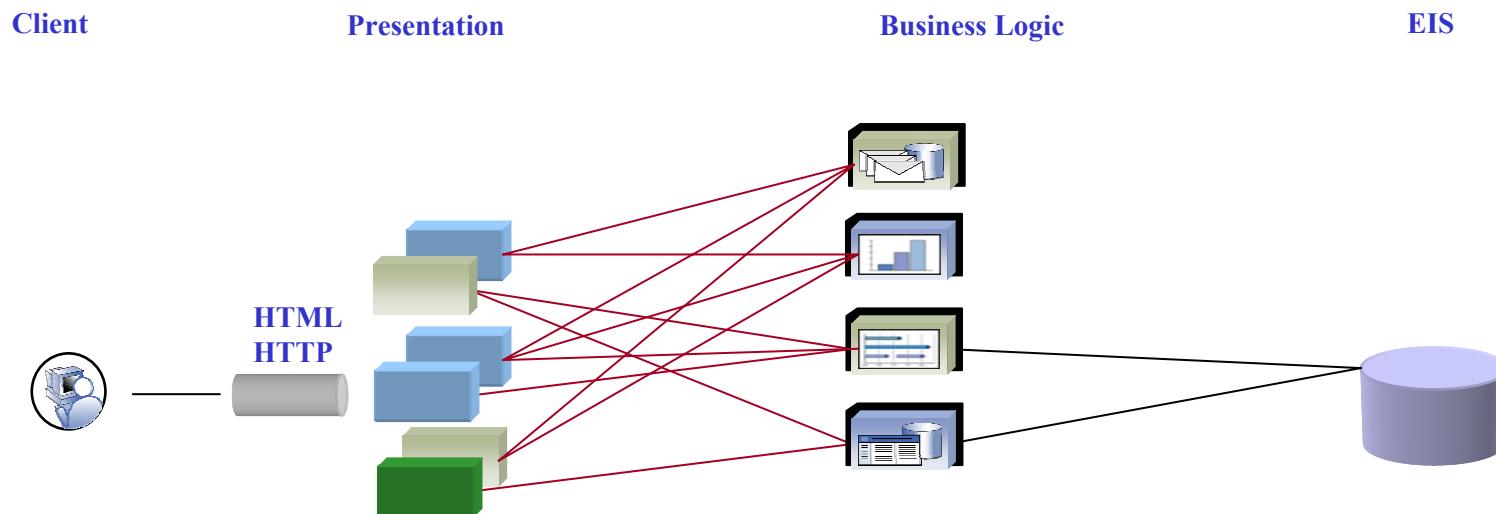
## 3-tier Architecture



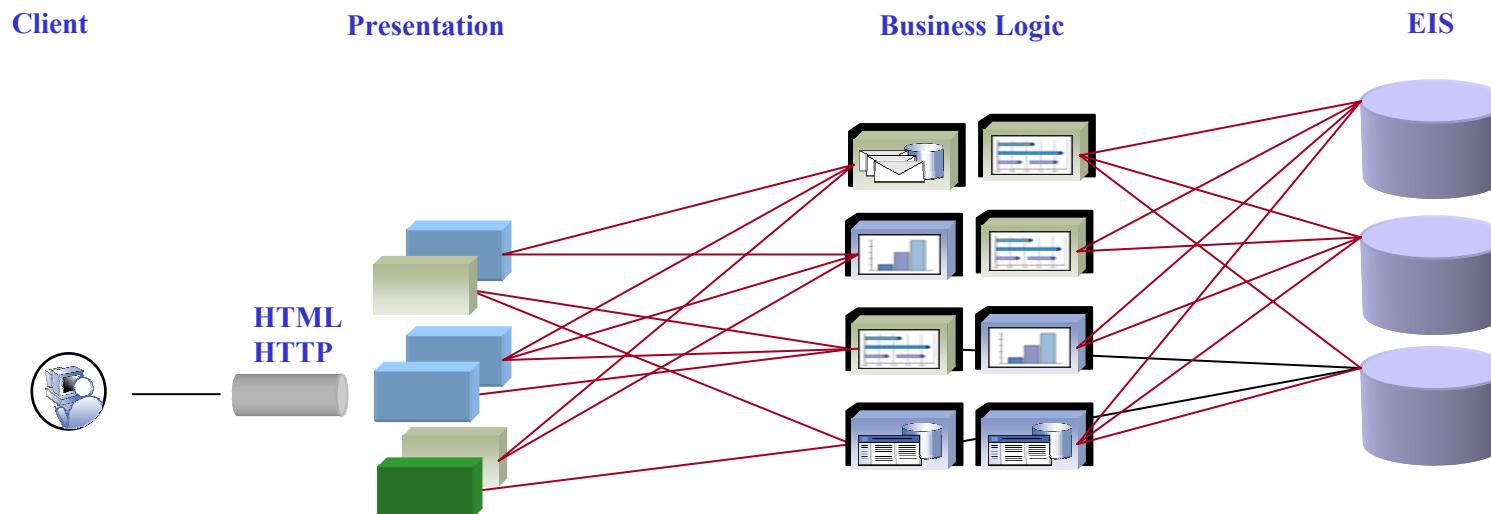
## 4-tier Architecture



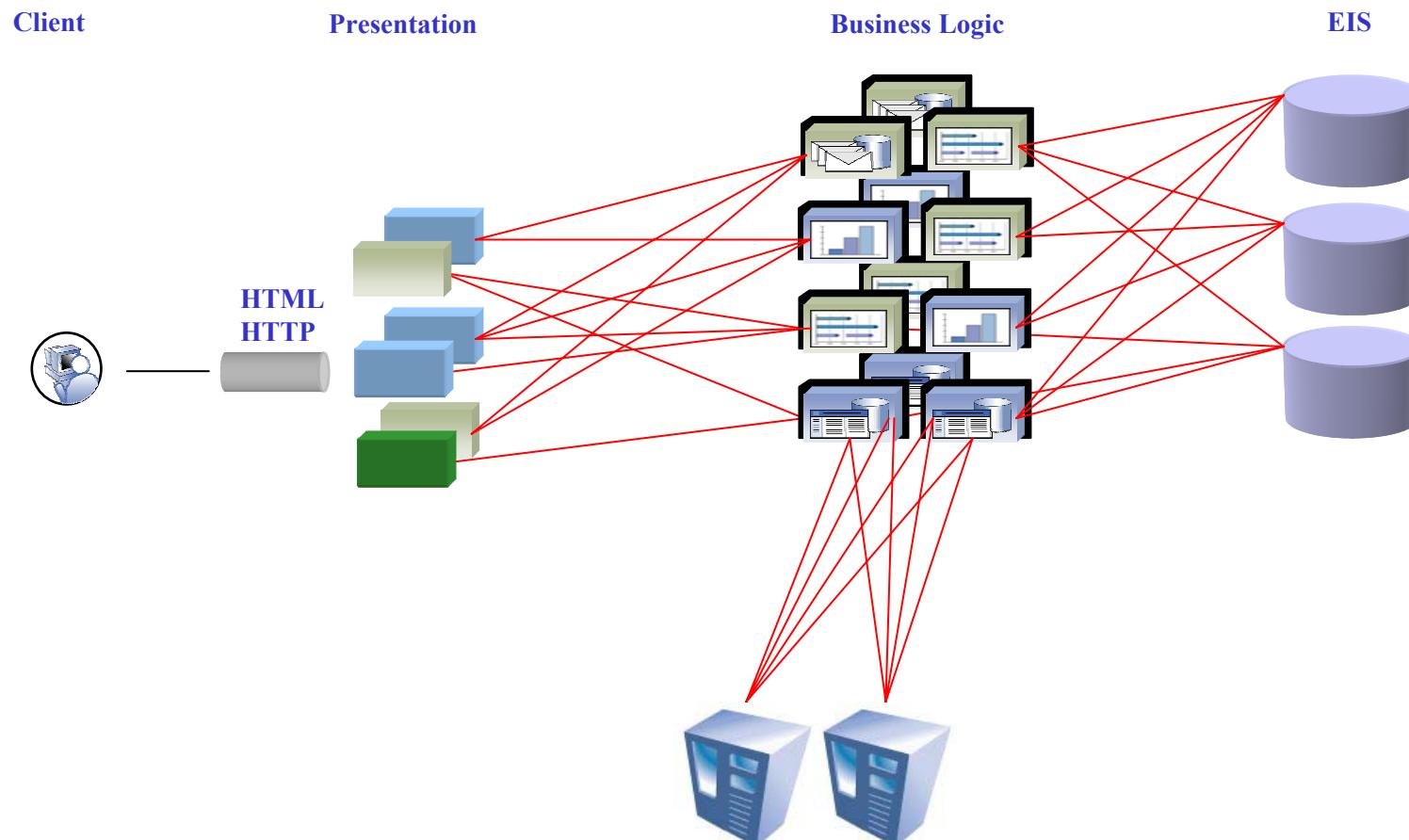
## 4-tier Architecture



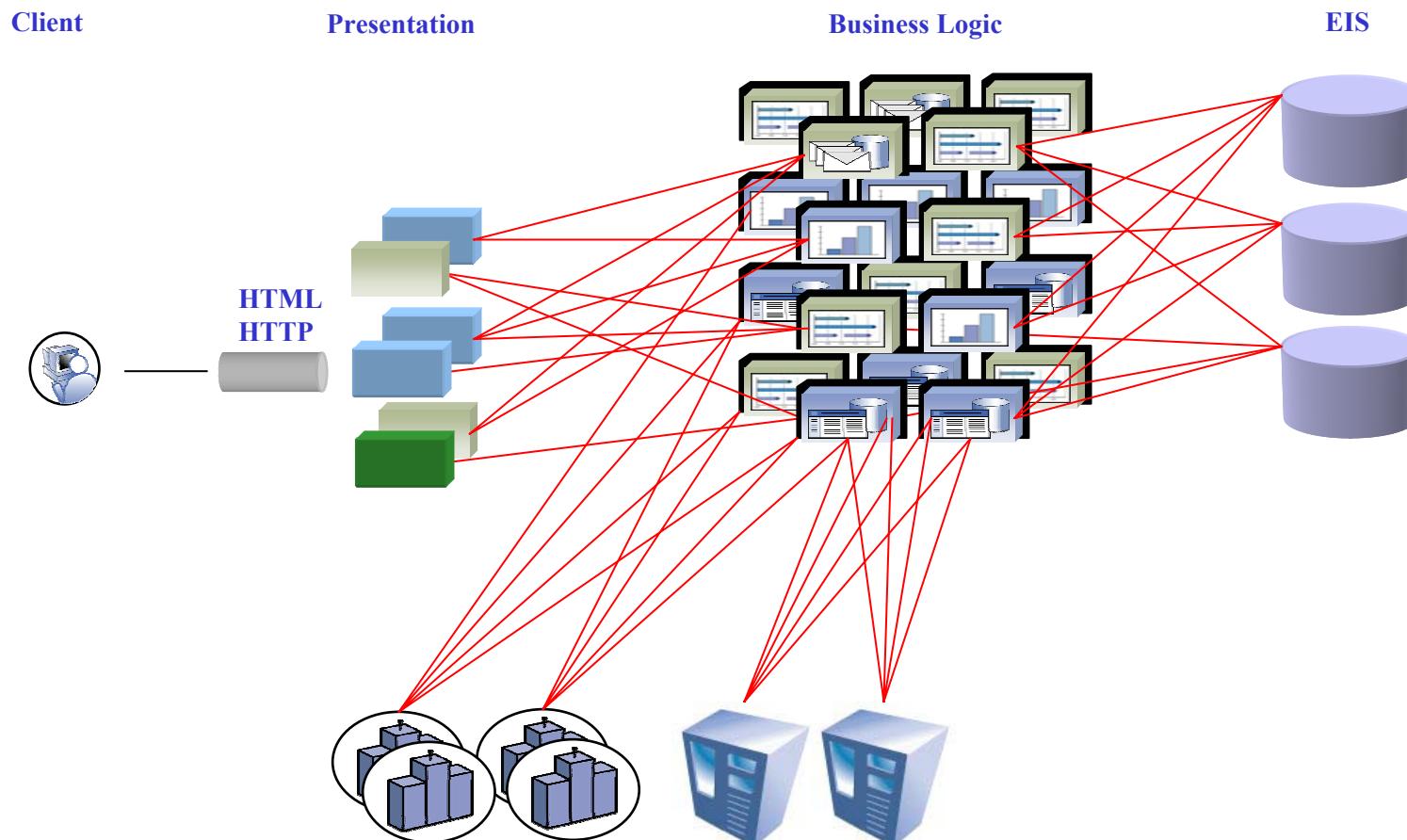
## 4-tier Architecture



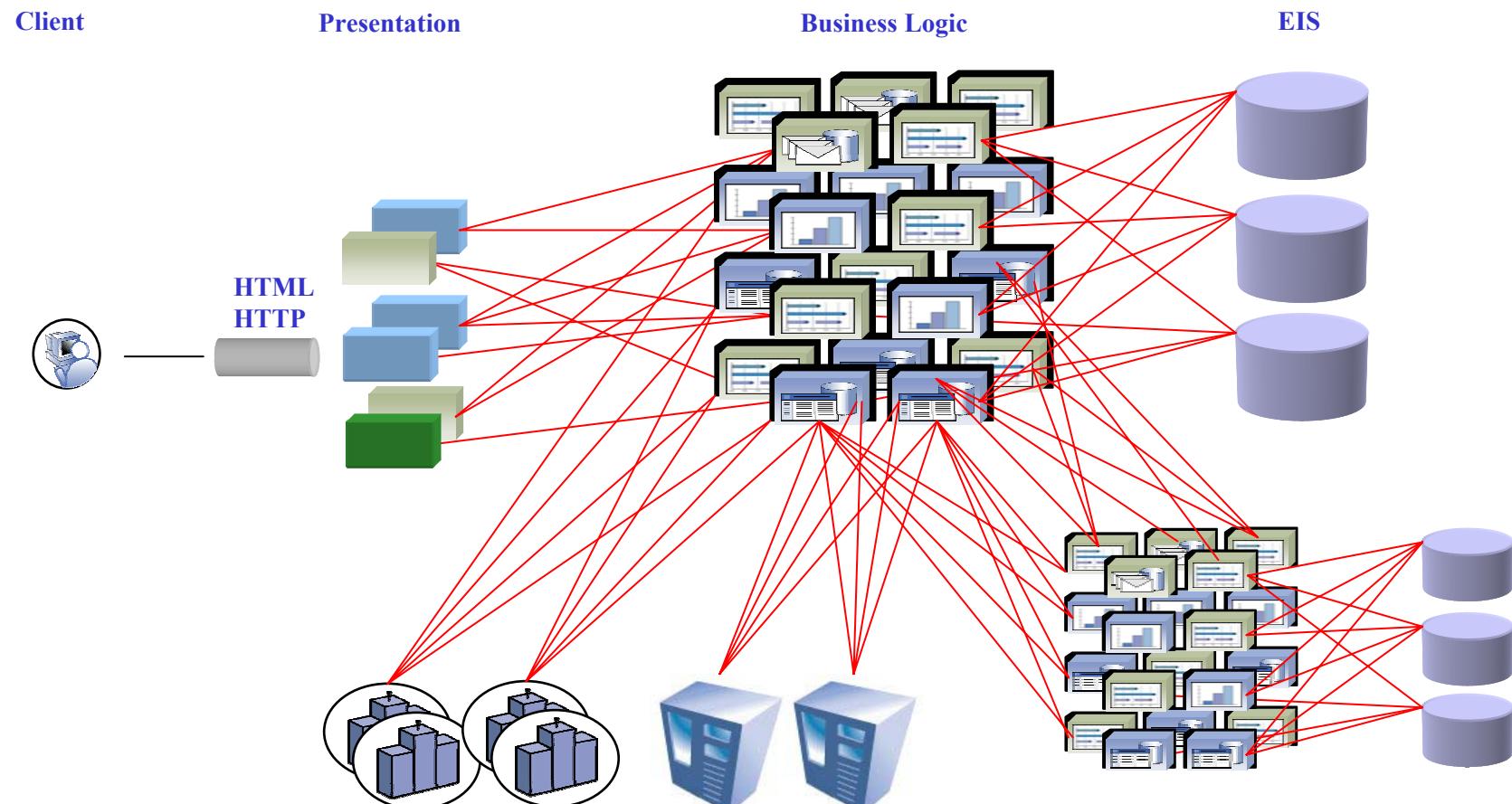
## 4-tier Architecture



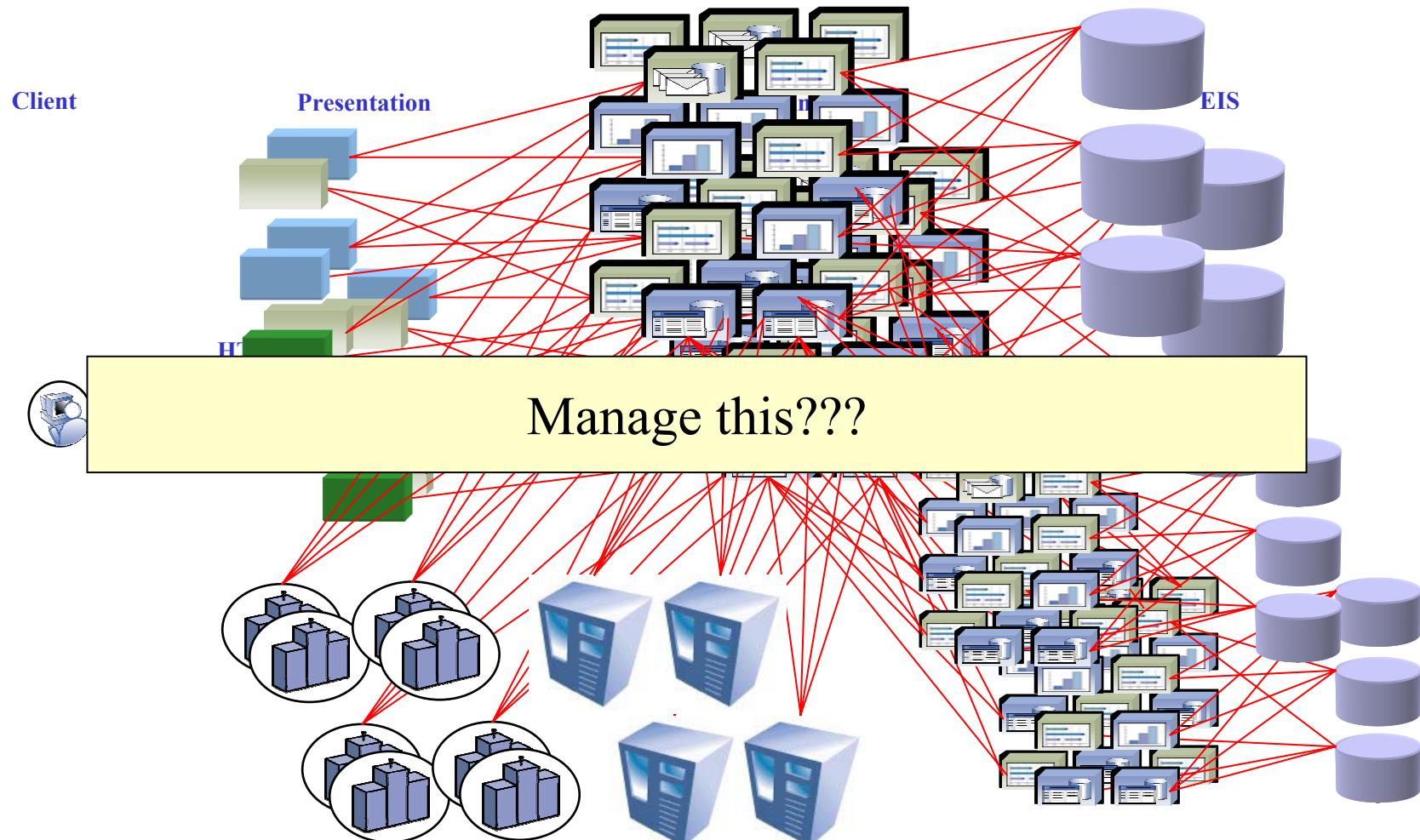
## 4-tier Architecture



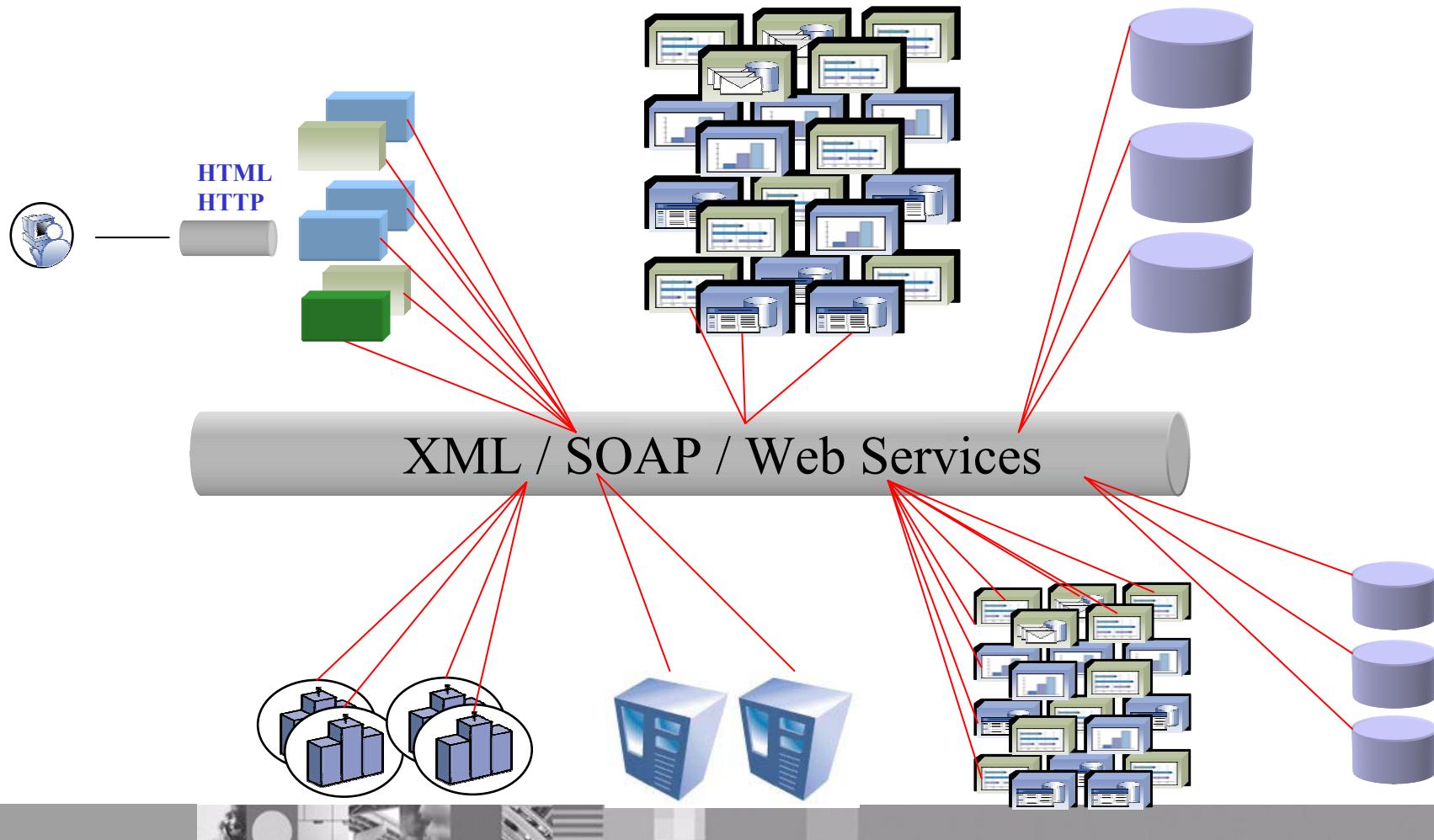
## 4-tier Architecture



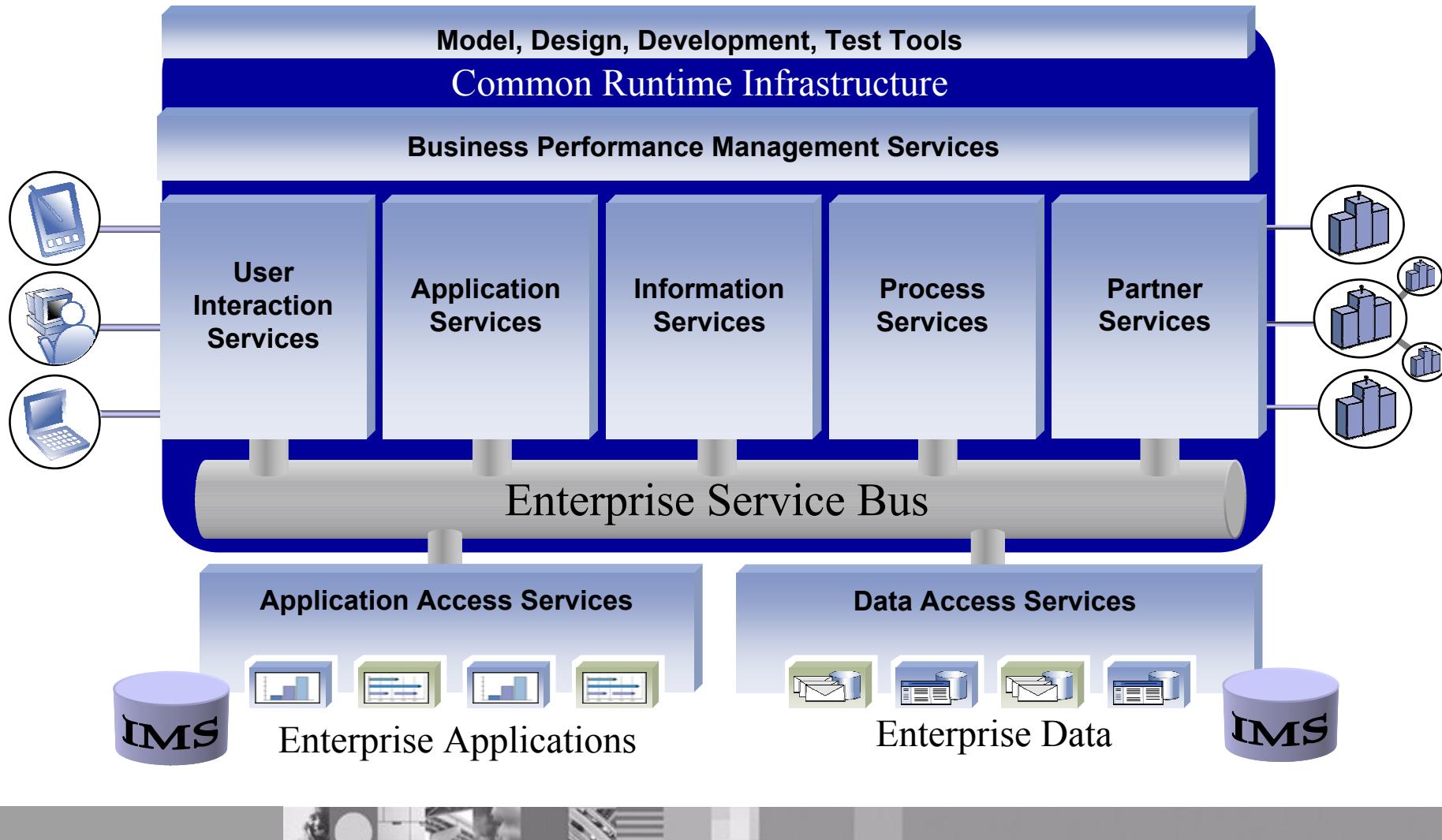
## 4-tier Architecture



## XML / SOAP / Web Services

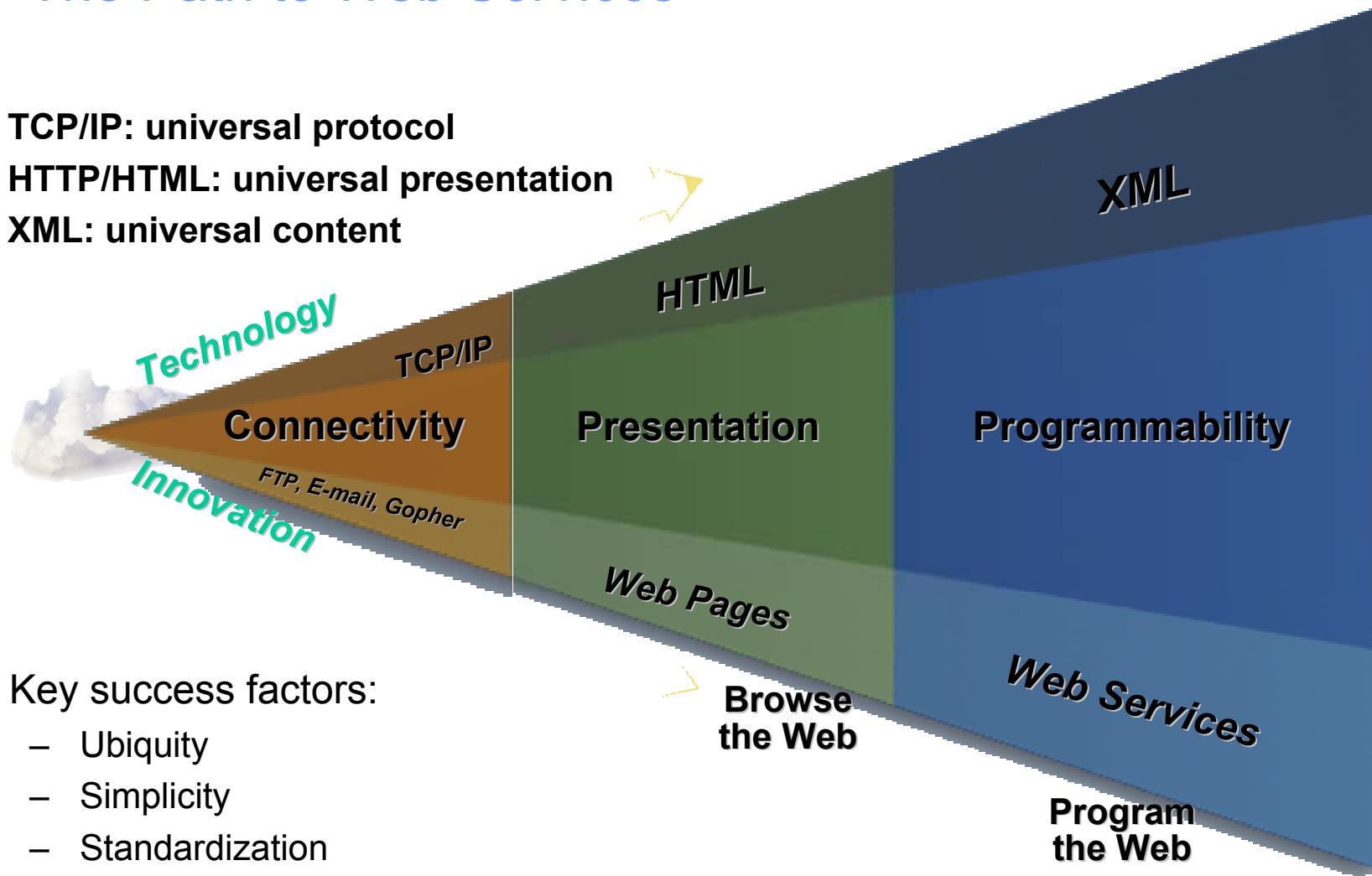


# IBM Service Oriented Architecture ESB Is a Defining Element



## The Path to Web Services

- TCP/IP: universal protocol
- HTTP/HTML: universal presentation
- XML: universal content



- Key success factors:
  - Ubiquity
  - Simplicity
  - Standardization

## What is IMS XML-DB

- New for IMS version 9.1
- A methodology for storing and retrieving XML documents into and out of standard IMS databases
  - Language Independent Design
  - XML Schema Metadata (Structural Metadata)
  - DL/I Metadata (Physical Metadata)
  - Two storage types
- Java enablement of XML-DB using an extended IMS Java JDBC interface





## Why use IMS XML-DB

- A World-wide movement towards XML as the standard data interchange language.
- Retrieve existing IMS data in standard, easily exchangeable XML format
- Store, Index, Search and Retrieve valid new XML documents into new or existing IMS databases
- 35 years of storage and management of Hierarchical data
- 35 years of performance, stability and reliability



## What is XML

- A Standardized, Simple, and Self-Describing Markup Language for documents containing structured or semi-structured information.

```
<?xml version="1.1"?>
<Presentation>
  <title>XML-DB</title>
  <length>60</length>
  <presenter>
    <lastName>Holtz</lastName>
    <firstName>Christopher</firstName>
  </presenter>
  <Comments session="e-business Design Review">
    <comment>Loved It</comment>
    <comment>Can't wait for GA</comment>
  </Comments>
</Presentation>
```



## Why XML...

- Standard Internet Data Exchange Format

- Self-Describing
  - Handles encoding

```
<xml? version="1.1" encoding="ebcdic-cp-us"?>
```

- Handles byte ordering

```
<OrderNumber>110203</OrderNumber>
```

- Human Legible?
  - Easily Parsed

- **Standard!**

## What is XML

- Data-centric
  - Highly structured
  - Limited size and strongly typed data elements
  - Order of elements generally insignificant
  - Invoices, purchase orders, etc.
- Document-centric
  - Loosely structured
  - Unpredictable sizes with mostly character data
  - Order of elements significant
  - Newspaper articles, manuals, etc.



## Well formed vs. Valid XML Document

- Well formed – Obeys the XML Syntax Rules
  - must begin with the XML declaration
  - must have one unique root element
  - all start tags must match end-tags
  - XML tags are case sensitive
  - all elements must be closed
  - all elements must be properly nested
  - all attribute values must be quoted
  - XML entities must be used for special characters
- Valid – Conforms to a specific XML Schema



## The XML Schema Definition Language

An XML language for defining the legal building blocks of a valid XML document

An XML Schema:

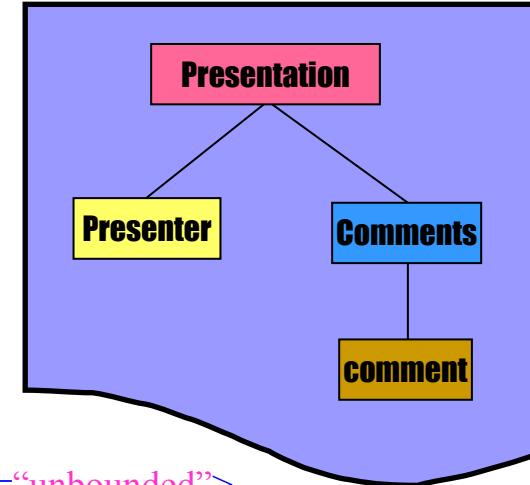
- defines elements and attributes that can appear in a document
- defines which elements are child elements
- defines the order and number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

Defines an agreed upon communication contract for exchanging XML documents

## XML Schema Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.myNamespace.net"
  targetNamespace="http://www.myNamespace.net"
  elementFormDefault="qualified">

  <xsd:element name="Presentation">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="length" type="xsd:integer"/>
        <xsd:element name="Presenter" minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="lastName" type="xsd:string"/>
          <xsd:element name="firstName" type="xsd:string"/>
        </xsd:element>
        <xsd:element name="Comments" minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="comment" minOccurs="0" maxOccurs="unbounded" type="xsd:string"/>
          <xsd:attribute name="session" type="xsd:string"/>
        ...
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



## XML Storage in IMS

- Natural mapping between hierarchic XML data and hierarchic IMS database definitions.

# **XML Schema**

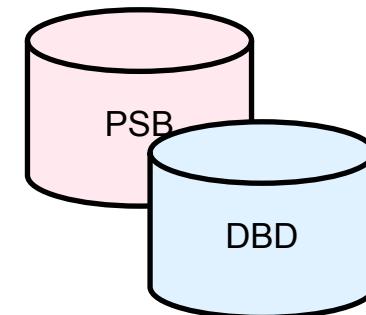
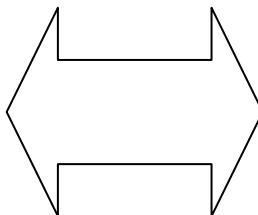
```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ims="http://www.ibm.com/ims"
  xmlns="http://www.ibm.com/ims/PSBName/PCBName"
  targetNamespace="http://www.ibm.com/ims/PSBName/PCBName"
  elementFormDefault="qualified">

  <xsd:annotation>
    <xsd:appinfo>
      <ims:DLI mode="store" PSB="AUTPSB11" PCB="AUTOLPCB"
        dsg="DATASETG" meanLength="1000" numDocs="100"/>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:schema>

<xsd:element name="A">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="field1" type="xsd:int"/>
      <xsd:element name="field2">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:maxLength value="30"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```



# It's the Metadata, Stupid!

- Physical Metadata
  - Segment Sizes
  - Segment Hierarchy (*field relationships – 1-to-1, 1-to-n*)
  - DBD Defined Fields
  - Application Defined Fields
  - Field Type, Type Length, Byte Ordering, Encoding, etc.
  - Offer Field/Segment Renaming (*lift 8 char restriction*)
- Structural Metadata
  - XML layout for fields (*field relationships must still match*)
  - Element vs. Attribute (*names must match*)
  - Type Restrictions, Enumerations, etc.

Defined in  
DBD

Defined in  
Copylibs  
(IMS Java)

Defined in  
XML Schema





# It's the Metadata, Stupid!

*Defined in  
DBD*

*Defined in  
Copylibs  
(IMS Java)*

## *Defined in XML Schema*

```
<PurchaseOrder number="113246">
    <lastName> Holtz      </lastName>
    <firstName> Christopher </firstName>
    <date>10/21/2003 </date>
    <payment type="MC"> 5414 2263 4895 1145 </payment>
```





*the world depends on it*

# Decomposed XML Retrieval in IMS

## Composed XML

```
<A>
<f1>  </f1>
<f2>  </f2>
<f3>  </f3>
<B>
<f4>  </f4>
<f5>  </f5>
</B>
<B>
<f4>  </f4>
<f5>  </f5>
</B>
<C>
<f6>  </f6>
<f7>  </f7>
<D>
<f8>  </f8>
<f9>  </f9>
</D>
</C>
</A>
```

Diagram illustrating XML Schema/Metadata:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ims="http://www.ibm.com/ims"
  xmlns="http://www.ibm.com/ims/PSBName/PCBName"
  targetNamespace="http://www.ibm.com/ims/PSBName/PCBName"
  elementFormDefault="qualified">

  <xsd:annotations go here/>

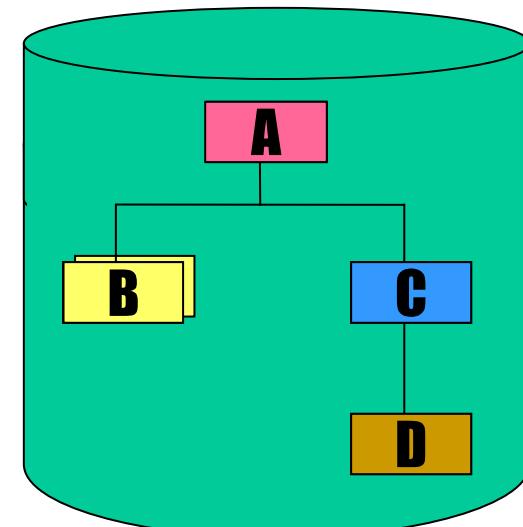
  <xsd:element name="A">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field1" type="xsd:int"/>
        <xsd:element name="field2">
          <xsd:simpleType>
            ...
          </xsd:simpleType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="B">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field4" type="xsd:int"/>
        <xsd:element name="field5" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name = "C">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field6" type="xsd:int"/>
        <xsd:element name="field7" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name = "D">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field8" type="xsd:int"/>
        <xsd:element name="field9" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

## XML Schema/ Metadata



## Decomposed Storage

- XML document must be parsed and validated.
- Data must be converted to *traditional* IMS types
  - COMP-1, COMP-2, etc.
  - EBCDIC CHAR, Picture Strings
- Stored data is searchable by IMS and transparently accessible by non-XML enabled applications.





*the world depends on it*

# Decomposed XML Storage in IMS

## Incoming XML

```
<A>
  <f1>  </f1>
  <f2>  </f2>
  <f3>  </f3>
<B>
  <f4>  </f4>
  <f5>  </f5>
</B>
<B>
  <f4>  </f4>
  <f5>  </f5>
</B>
<C>
  <f6>  </f6>
  <f7>  </f7>
<D>
  <f8>  </f8>
  <f9>  </f9>
</D>
</C>
</A>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ims="http://www.ibm.com/ims"
  xmlns="http://www.ibm.com/ims/PSBName/PCBName"
  targetNamespace="http://www.ibm.com/ims/PSBName/PCBName"
  elementFormDefault="qualified">

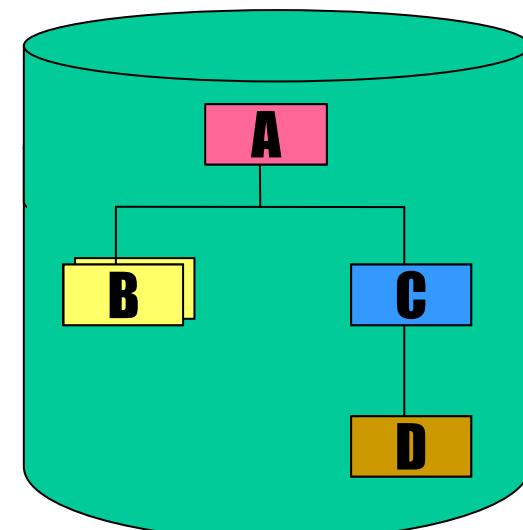
  <xsd:element name="A">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field1" type="xsd:int"/>
        <xsd:element name="field2">
          <xsd:simpleType>
            ...
          </xsd:simpleType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="B">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field3" type="xsd:int"/>
        <xsd:element name="field4">
          <xsd:simpleType>
            ...
          </xsd:simpleType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name = "C">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field5" type="xsd:int"/>
        <xsd:element name="field6">
          <xsd:simpleType>
            ...
          </xsd:simpleType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name = "D">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field7" type="xsd:int"/>
        <xsd:element name="field8">
          <xsd:simpleType>
            ...
          </xsd:simpleType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

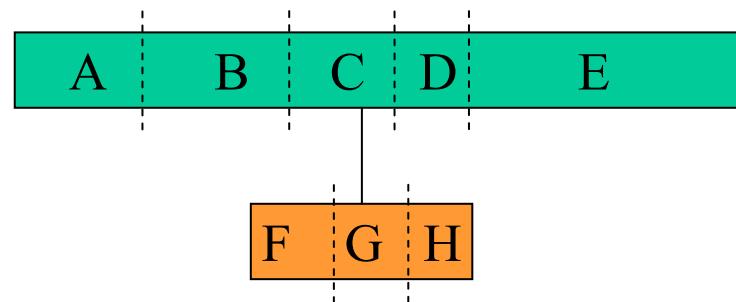
## XML Schema/ Metadata





## Simple XML Structuring

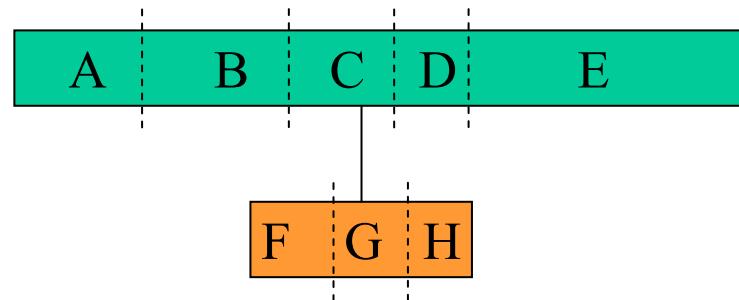
- Uses Default structure
- All data is represented as elements



```
<?xml version="1.0"?>
<SegName>
  <A>[A]</A>
  <B>[B]</B>
  <C>[C]</C>
  <D>[D]</D>
  <E>[E]</E>
<Seg2Name>
  <F>[F]</F>
  <G>[G]</G>
  <H>[H]</H>
</Seg2Name>
<Seg2Name>
  <F>[F]</F>
  <G>[G]</G>
  ...
</SegName>
```

## Complex XML Structuring

- Uses Attribute Mapping
- Uses hard coded values
- Uses hard coded levels



```
<?xml version="1.0"?>
<document A="[ ]" B="[ ]">
  <G F="[ ]" H="[ ]">[ ]</G>
  <G F="[ ]" H="[ ]">[ ]</G>
  ...
  <sub C="[ ]">
    <another attr="38">
      <E D="[ ]">[ ]</E>
    </sub>
  </document>
```

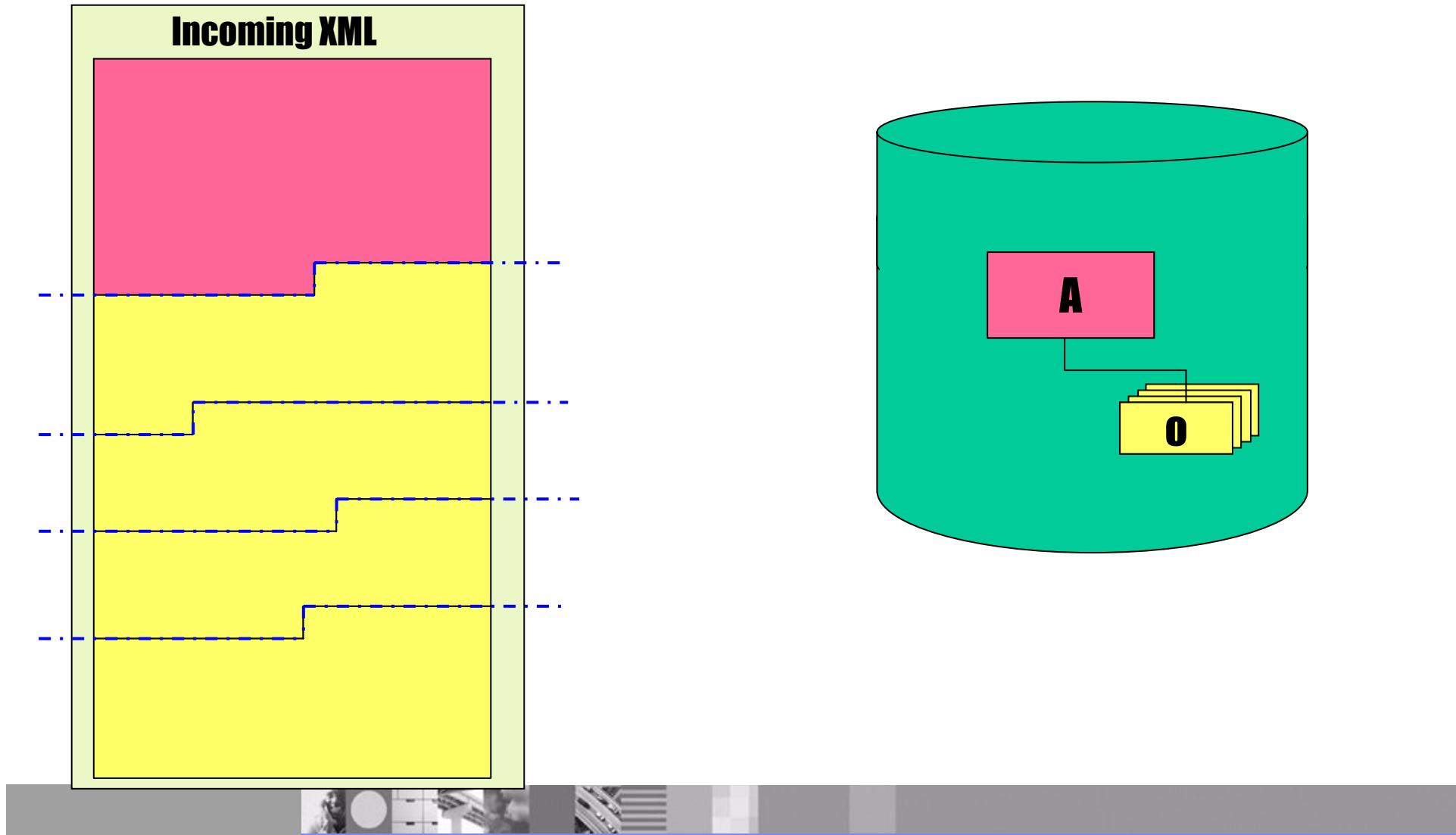
## Intact Storage

- No (or little) XML Parsing or Schema validation
  - Storage and Retrieval Performance
- No (or little) data type conversions
  - Unicode storage
- Stored documents are no longer searchable by IMS and only accessible to XML-enabled applications
  - XPath side segments



*the world depends on it*

## Intact XML Storage in IMS



## Intact Storage Structure

- Base Segment

- 1 byte
- 1 byte
- 2 bytes
- n bytes

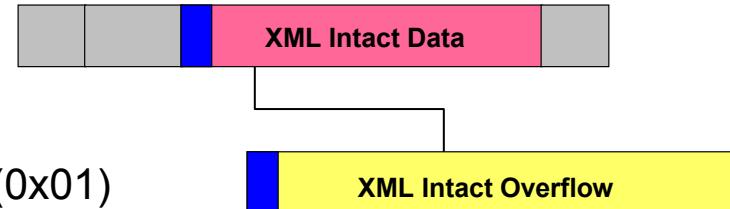
Version Number (0x01)

Reserved

1<sup>st</sup> bit: indicates more segments than just this one

2<sup>nd</sup>–16<sup>th</sup> bits: indicates this segments length

Intact XML Data



- Overflow Segment

- 2 bytes
- 2 bytes
- n bytes

Key field sequence number

1<sup>st</sup> bit: indicates more segments than just this one

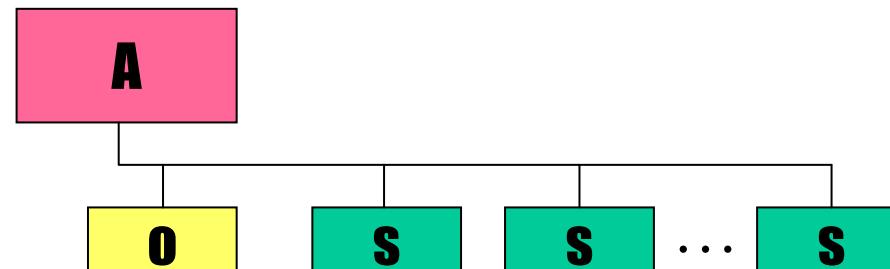
2<sup>nd</sup>–16<sup>th</sup> bits: indicates this segments length

Intact XML Data

## Intact Storage Secondary Indexing

- XPath expression identifying Side Segments
  - Side segment is converted to *traditional* data type and copied into segment.
- Side Segments are secondary indexed with documents root as target.

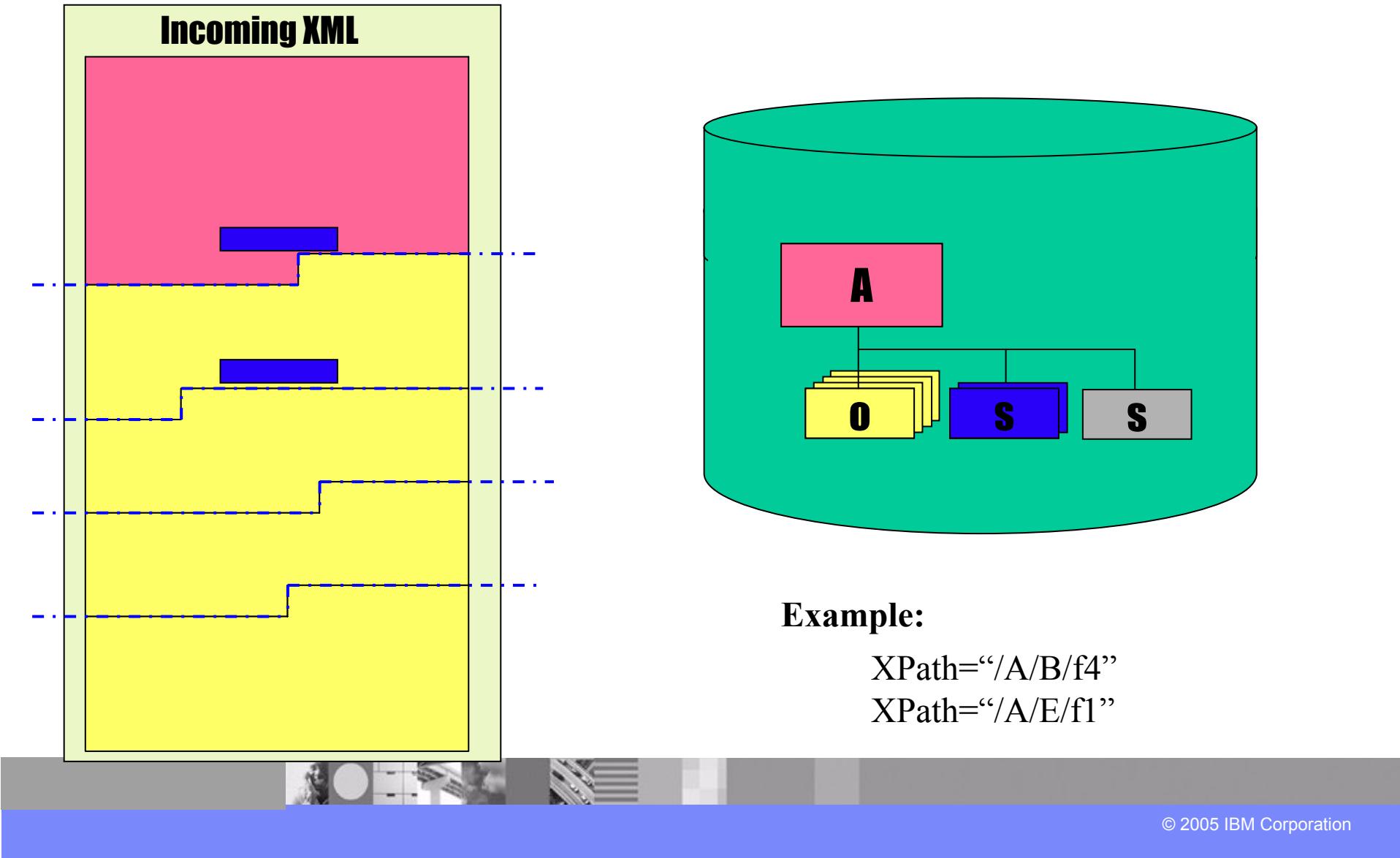
Example:



XPath="/Dealer/DealerName"

XPath="/Dealer/Model[Year>1995]/Order/LastName"

## Intact XML Storage in IMS



## Decomposed vs. Intact Storage

- **Decomposed** (*data-centric storage*)
  - XML tags are stripped from XML data
  - Identical as current IMS storage
  - Strict data-centric XML Schema validated data
  - EBCDIC encoding
  - Searching on IMS Search Fields
- **Intact** (*document-centric storage*)
  - Entire XML document is stored (including tags)
  - Relaxed un-validated data
  - Any desired encoding is possible
  - Searching is through XPath specified and generated Secondary Indexed Side Segments



## Overview

- Introduction
  - What is IMS XML-DB?
  - What is XML?
  - What are XML Schemas?
- IMS XML-DB Methodology (XML Visualization)
  - Decomposed Storage
  - Intact Storage
- IMS Java / XML-DB Tooling
  - Metadata Generation
- IMS XML-DB Java Implementation
  - SQL UDF Interface
  - Future Direction

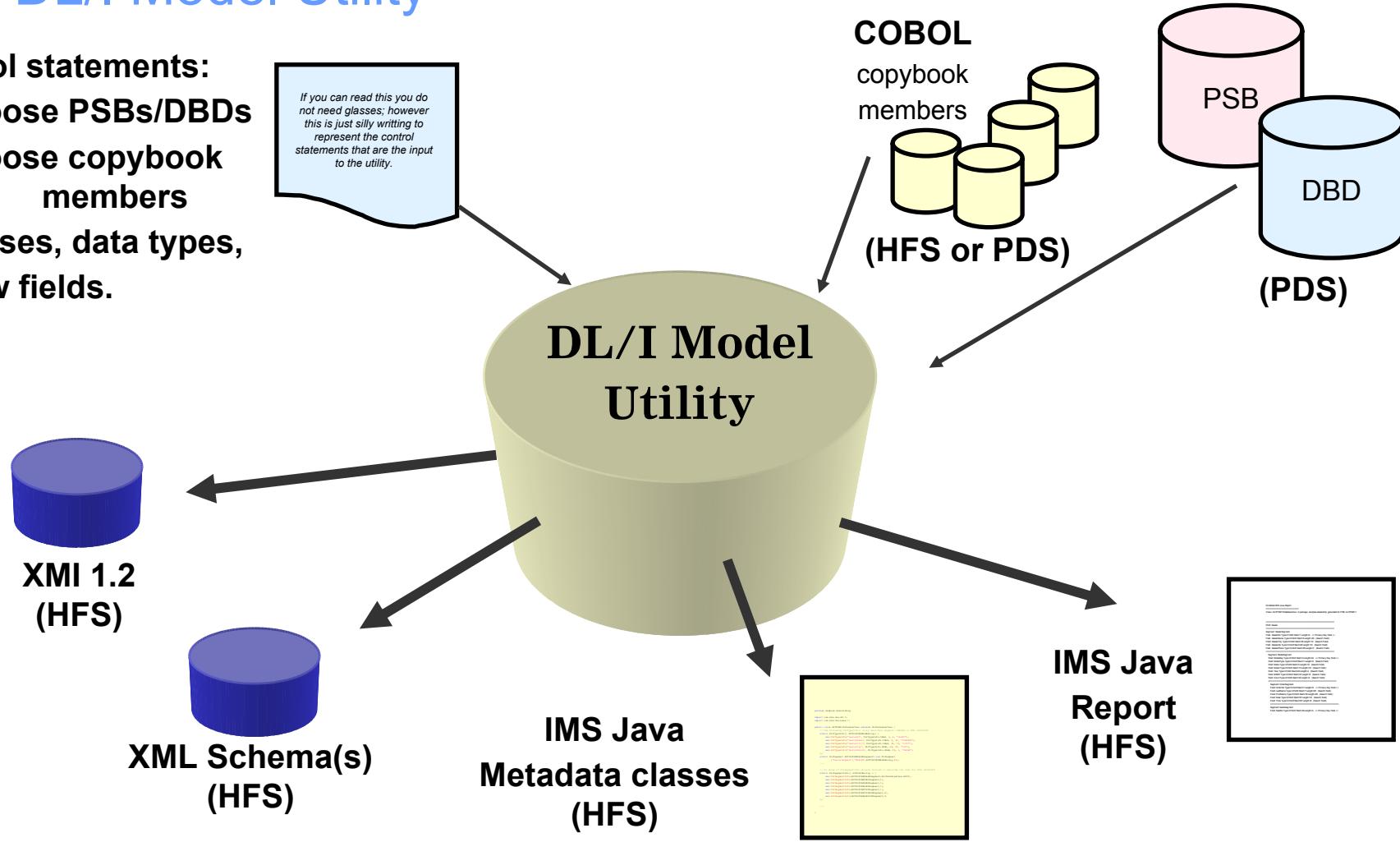


## DL/I Model Utility

**Control statements:**

- 1) Choose PSBs/DBDs
- 2) Choose copybook members
- 3) Aliases, data types, new fields.

If you can read this you do not need glasses; however this is just silly writing to represent the control statements that are the input to the utility.



## DL/I Model Schema Generation

- Additional Control Statements Keywords

```
OPTIONS PSBds=PSB.SOURCE.PDS      DBDds=DBD.SOURCE.PDS
       GenJavaSource=YES           JavaSourcePath=output/dir
       Package=test.db.psb4        ReportPath=output/dir
GenXMLSchemas=store           XMLSchemaPath=output/dir
       Outpath=output/dir
```

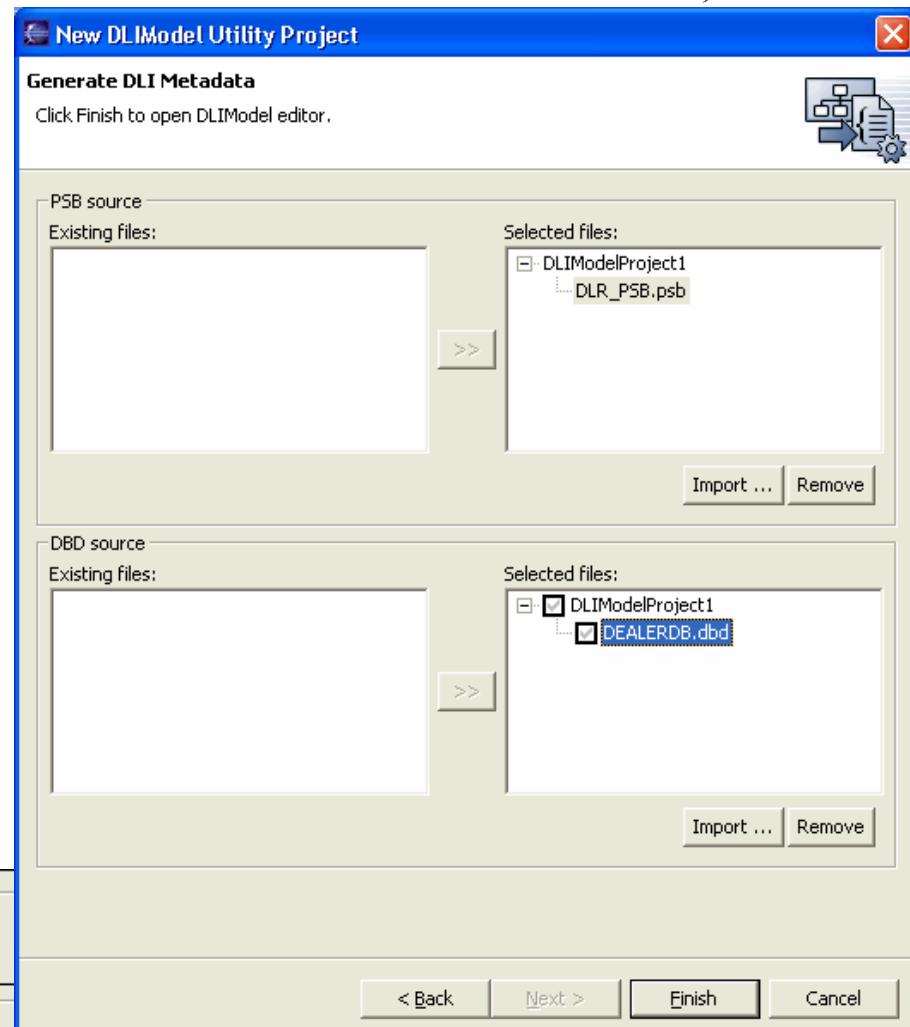
```
PSB psbName=PREPSB4 Javaname=PresentationDatabase
    PCB PCBName=PCB1 JavaName=FullView GenXMLSchema=store
```

```
// Physical Segments for PREZDB
SEGMENT DBDName=PREZDB SegmentName=PRSNTON JavaName=Presentation
  FIELD Name=TITLE   JavaType=CHAR   JavaName=title
  FIELD Name=LENGTH JavaType=INTEGER JavaName=length
  ...
  ...
```



## GUI DL/I Model Utility

- Select DBDs / PSBs
  - must be FTPed locally
- Source is Parsed
  - any errors are reported
- XMI Metamodel
  - generated
  - opened for editing



*the world depends on it*

# DL/I Model GUI Editor

DLR\_PSB.java

Edit metadata of 'DLR\_PSB.java' [project: DLIModelProject1/DLR\_PSB]

**Metadata View**

PSB > PCB > segment tree

- DLR\_PSB
  - DLR\_PCB1
    - DEALER
      - MODEL
        - ORDER1
        - SALES
        - STOCK

Segment > field tree [start - end]

Field	Start	End
MODEL	[1 - 43]	
MODTYPE	[1 - 21]	
MAKE	[3 - 12]	
MODEL	[13 - 22]	
YEAR	[23 - 26]	
MSRP	[27 - 31]	

**Metadata was generated.**

You can now exit the DLIModel editor and use the metadata to write application, or you can modify the metadata and create an XML schema before you exit the editor.

The metadata was saved in the following project folder:  
C:/Alison\_Docs/DLIModel/eclipse-SDK-2.1/eclipse-SDK-2.1.3-w

**Modify Metadata**

You can modify the metadata by importing COBOL XMI or manually modifying the field information.

Import COBOL XMI to enhance metadata with a COBOL copybook.

**Import COBOL XMI**

Select a field or segment from the field tree to modify it.

**Delete Field** **Edit...** **Add Field**

Reset metadata to its initial state.

**Reset Metadata**

**Create XML Schema**

You can create an XML schema to validate XML documents that are retrieved from or stored in IMS.

Select a metadata root element from the segment tree.

MODEL

**Create Schema**

**Trace Log**

Write a DLIModel Utility trace log (dlimodeltrace.txt)

DLI Editor DLIDatabaseView



# GUI DL/I Model Utility – Physical Metadata

The screenshot shows the Eclipse Platform interface with the title "Java - DLR\_PSB.java - Eclipse Platform". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations. The left pane is the "Package Explorer" showing the project structure:

- DLIModelProject1
  - bin
  - DLR\_PSB
    - DLR\_PSB.java
    - DLR\_PSB
      - DLR\_PCB1Array
      - DLR\_PCB1DEALERArray
      - DLR\_PCB1DEALERSegment
      - DLR\_PCB1MODELArray
      - DLR\_PCB1MODELSegment
      - DLR\_PCB1ORDERArray
      - DLR\_PCB1ORDERSegment
      - DLR\_PCB1SALESArray
      - DLR\_PCB1SALESSegment
      - DLR\_PCB1STOCKArray
      - DLR\_PCB1STOCKSegment
      - DLR\_PSB
  - JRE System Library [Java141]
  - imsjava.jar - C:\Alison\_Docs\DLIModel\ed
  - DBD Resources
    - DEALERDB.dbd
  - PSB Resources
    - DLR\_PSB.psb

```
package DLR_PSB;

import com.ibm.ims.db.*;
import com.ibm.ims.base.*;

public class DLR_PSB extends DLIDatabaseView {

    // This class describes the data view of PSB: DLR_PSB
    // PSB DLR_PSB has database PCBs with 8-char PCBNAME or label:
    //     DLR_PCB1

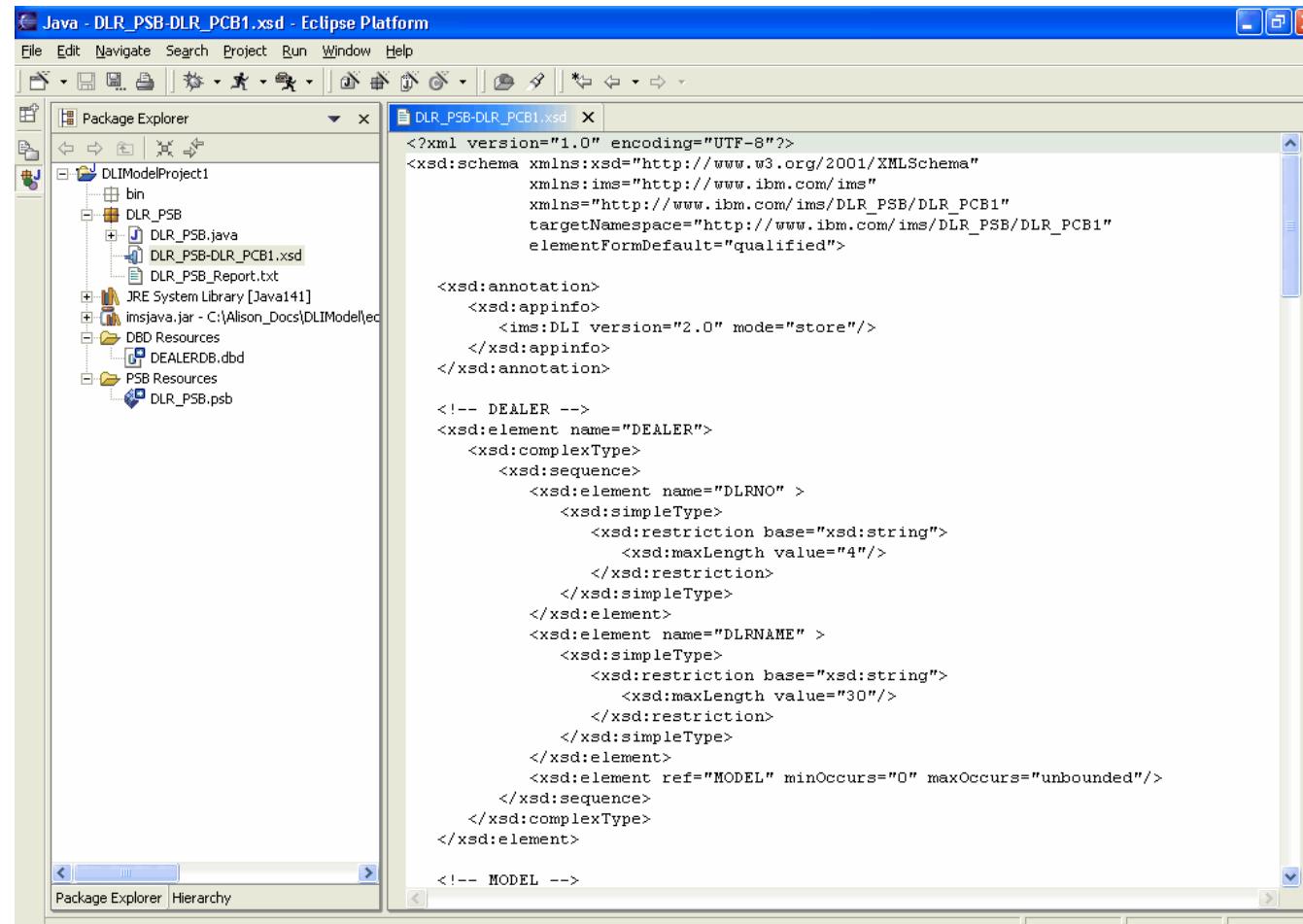
    // The following describes Segment: DEALER ("DEALER") in PCB: DLR_PCB1 ("DLR_PCB1")
    static DLITypeInfo[] DLR_PCB1DEALERArray= {
        new DLITypeInfo("DLRNO", DLITypeInfo.CHAR, 1, 4, "DLRNO", DLITypeInfo.UNIQUE),
        new DLITypeInfo("DLRNAME", DLITypeInfo.CHAR, 5, 30, "DLRNAME")
    };
    static DLISegment DLR_PCB1DEALERSegment= new DLISegment
        ("DEALER", "DEALER", DLR_PCB1DEALERArray, 94);

    // The following describes Segment: MODEL ("MODEL") in PCB: DLR_PCB1 ("DLR_PCB1")
    static DLITypeInfo[] DLR_PCB1MODELArray= {
        new DLITypeInfo("MODTYPE", DLITypeInfo.CHAR, 1, 2, "MODTYPE", DLITypeInfo.UNIQUE),
        new DLITypeInfo("MAKE", DLITypeInfo.CHAR, 3, 10, "MAKE"),
        new DLITypeInfo("MODEL", DLITypeInfo.CHAR, 13, 10, "MODEL"),
        new DLITypeInfo("YEAR", DLITypeInfo.CHAR, 23, 4, "YEAR"),
        new DLITypeInfo("MSRP", DLITypeInfo.CHAR, 27, 5, "MSRP")
    };
    static DLISegment DLR_PCB1MODELSegment= new DLISegment
        ("MODEL", "MODEL", DLR_PCB1MODELArray, 43);

    // The following describes Segment: ORDER ("ORDER1") in PCB: DLR_PCB1 ("DLR_PCB1")
    static DLITypeInfo[] DLR_PCB1ORDERArray= {
        new DLITypeInfo("ORDNBR", DLITypeInfo.CHAR, 1, 6, "ORDNBR", DLITypeInfo.UNIQUE),
        new DLITypeInfo("LASTNME", DLITypeInfo.CHAR, 50, 25, "LASTNME"),
        new DLITypeInfo("FIRSTNME", DLITypeInfo.CHAR, 75, 25, "FIRSTNME")
    };
    static DLISegment DLR_PCB1ORDERSegment= new DLISegment
```

Writable Insert 1:1

# GUI DL/I Model Utility – Structural Metadata



```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ims="http://www.ibm.com/ims"
  xmlns="http://www.ibm.com/ims/DLR_PSB/DLR_PCB1"
  targetNamespace="http://www.ibm.com/ims/DLR_PSB/DLR_PCB1"
  elementFormDefault="qualified">

  <xsd:annotation>
    <xsd:appinfo>
      <ims:DLI version="2.0" mode="store"/>
    </xsd:appinfo>
  </xsd:annotation>

  <!-- DEALER -->
  <xsd:element name="DEALER">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="DLRNO" >
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:maxLength value="4"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="DLRNAME" >
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:maxLength value="30"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element ref="MODEL" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <!-- MODEL -->
```

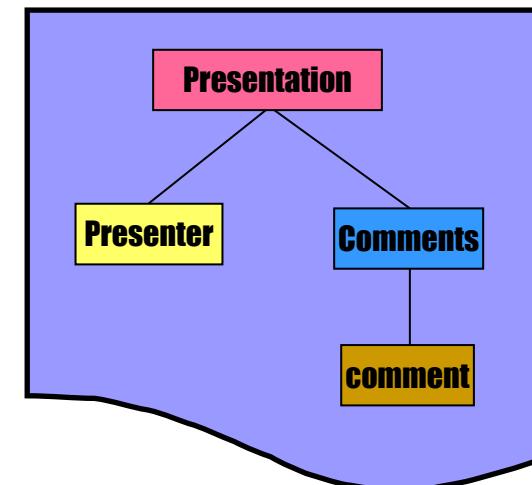
*the world depends on it*

## XML Schema Example

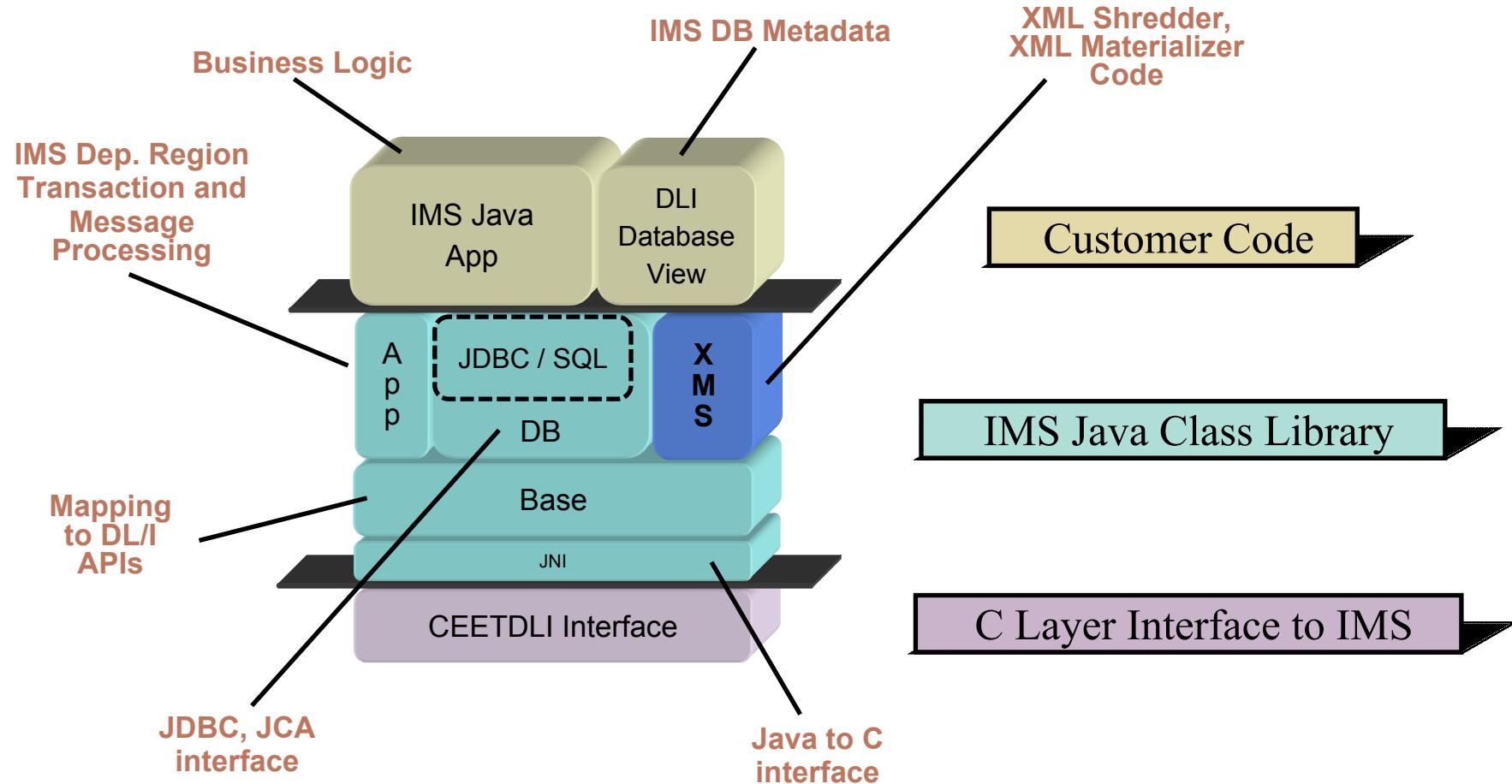
```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.ibm.com/ims/PREPSB4/FullView"
    targetNamespace="http://www.ibm.com/ims/PREPSB4/FullView"
    elementFormDefault="qualified">

    <xsd:annotation>
        <xsd:appinfo>
            <ims:DLI mode="store"/>
        </xsd:appinfo>
    </xsd:annotation>

    <xsd:element name="Presentation">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="title" type="xsd:string"/>
                <xsd:element name="length" type="xsd:integer"/>
                <xsd:element ref="Presenter" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="Comments" minOccurs="0" maxOccurs="unbounded"/>
            ...
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```



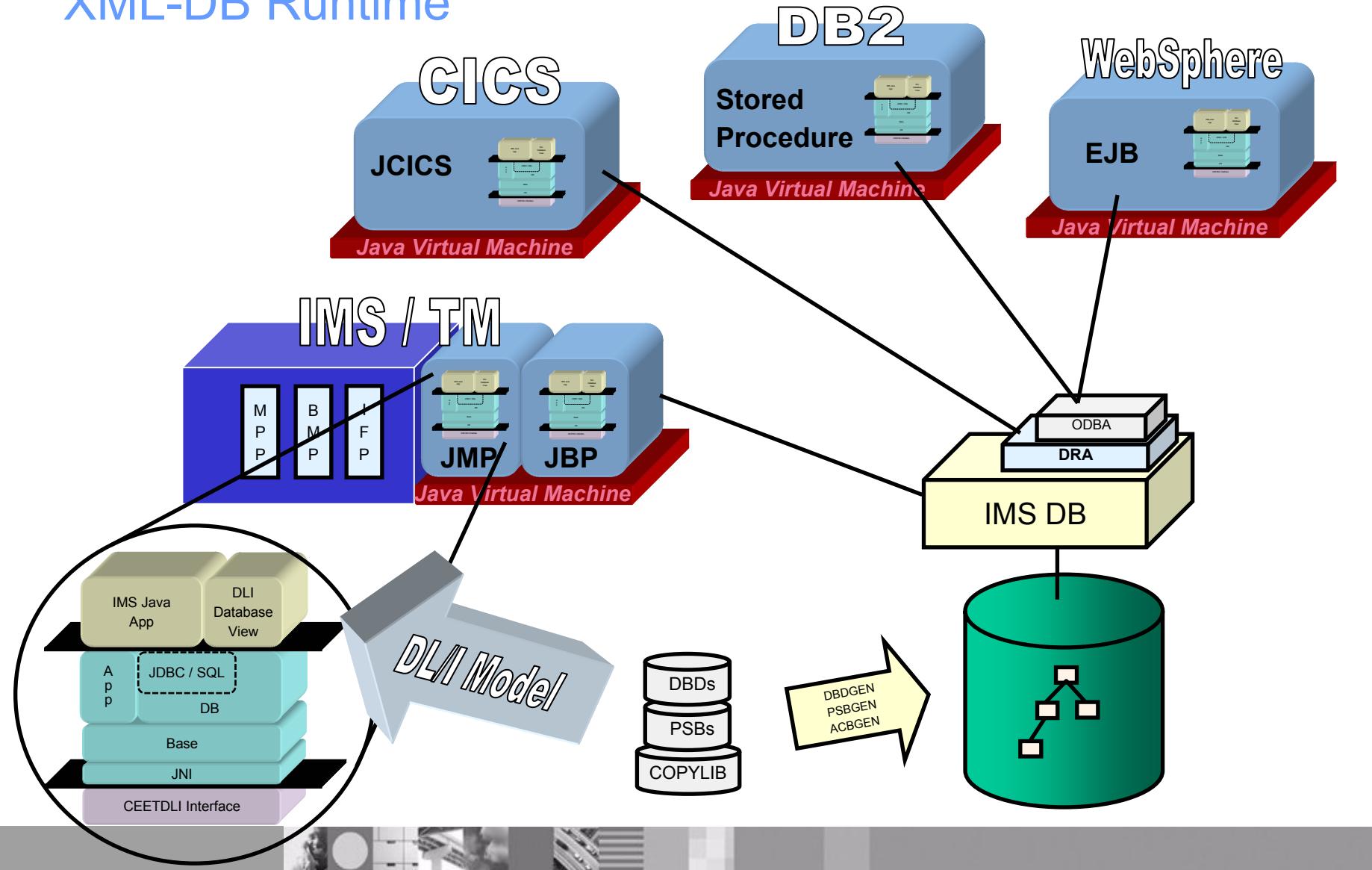
# IMS XML-DB Java Implementation





*the world depends on it*

## XML-DB Runtime



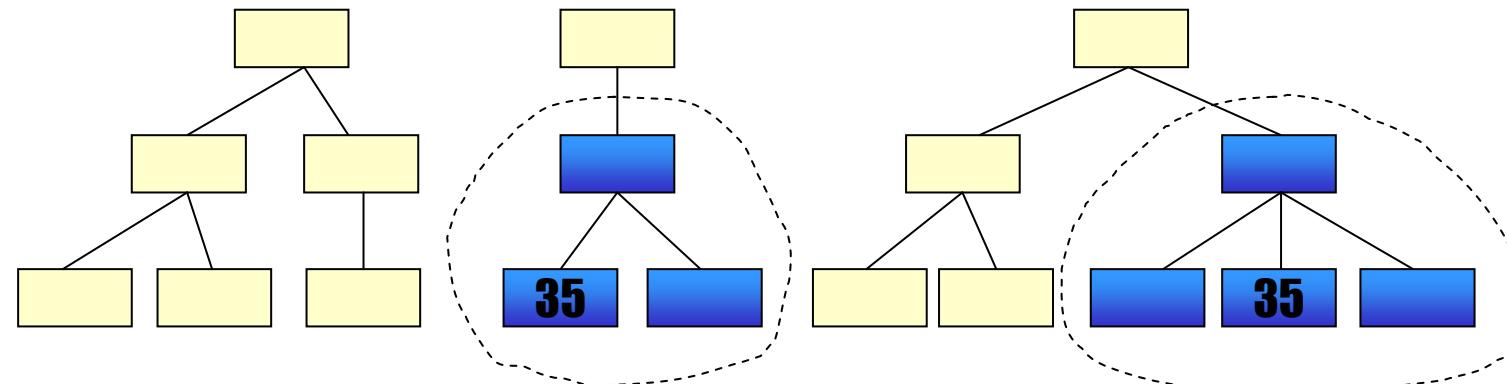
## IMS XML-DB Java Interface

- Adds 2 User Defined Functions (UDF) to the IMS Java JDBC SQL interface
  - retrieveXML()
  - storeXML()
- Runs as an IMS Java Application
  - JDR (JMP, JBP) *(IMS TM 9.1 or higher)*
  - DB2 Stored Procedure *(DB2 v7 or higher)*
  - CICS *(CICS TS 2.2 or higher)*
  - WebSphere *(5.01 or higher)*



## RetrieveXML() UDF

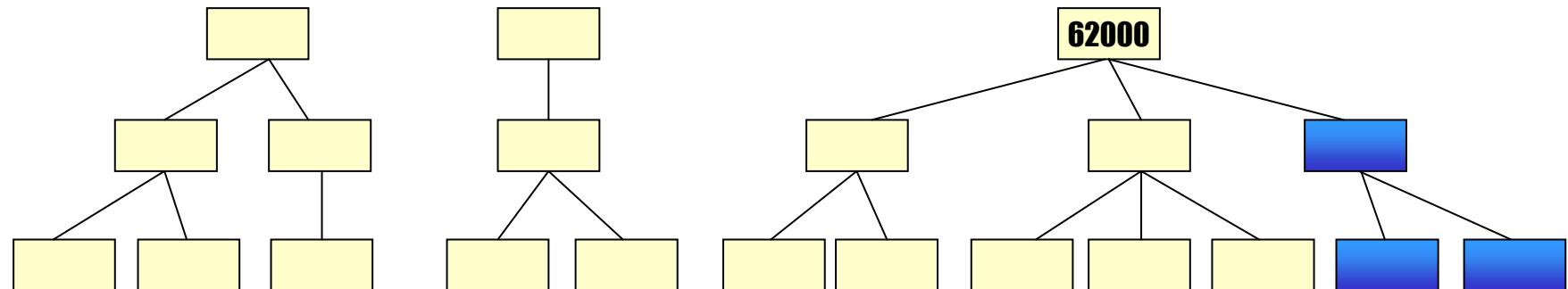
```
SELECT retrieveXML(B)
FROM C
WHERE C.fieldA = '35'
```



\*Two Rows of XML CLOBs in the ResultSet

## StoreXML() UDF

```
INSERT INTO B (storeXML())
VALUES (?)
WHERE A.fieldA = '62000'
```



\*Insert Statement must be a Prepared Statement

## Execute Query

retrieveXML() call

```
public void processMessage(String dealerName) {  
    obtain connection...  
  
    String query =  
        "SELECT DealerSegment.DealerName, retrieveXML(DealerSegment) AS DealerXMLDoc" +  
        " FROM Dealer.DealerSegment" +  
        " WHERE DealerSegment.DealerName = '" + dealerName + "'";  
  
    Statement statement = connection.createStatement();  
    ResultSet results = statement.executeQuery(query);  
  
    process results...  
    close connection...  
}
```

*the world depends on it*

## Process Results

getBlob() call

```
public void processMessage(String dealerName) {  
    obtain connection...  
    execute query...  
  
    while (results.next()) {  
        Clob xmlDoc = results.getBlob("DealerXMLDoc");  
  
        saveBlobToFile(xmlDoc, results.getString("DealerName"));  
    }  
  
    close connection...  
}
```



## Process Results

getCharacterStream() or  
getAsciiStream()

```
public void saveClobToFile(Clob clob, String fileName) throws IOException {  
  
    Reader reader = clob.getCharacterStream();  
    FileWriter writer = new FileWriter(fileName + ".xml");  
  
    char[] line = new char[1024];  
    int x = reader.read(line,0,1024);  
    while (x != -1) {  
        writer.write(line,0,x);  
        x = reader.read(line,0,1024);  
    }  
  
    reader.close();  
    writer.close();  
}
```



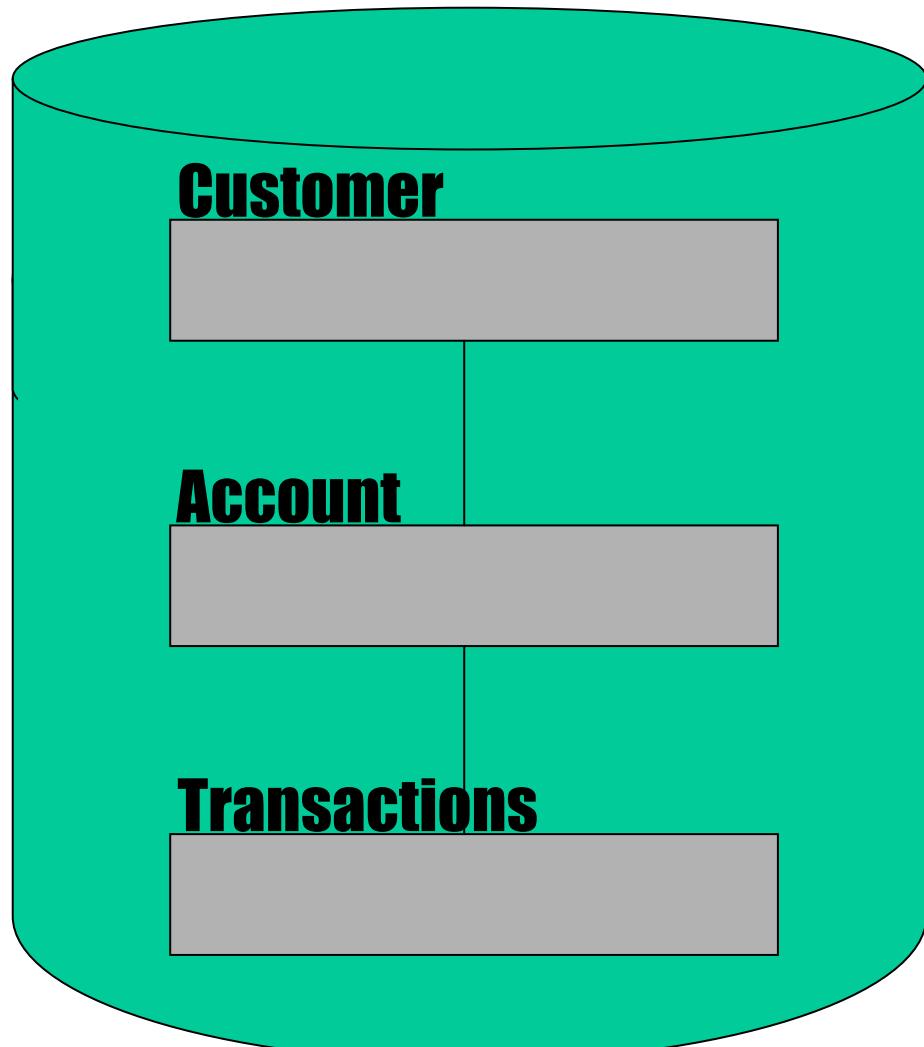
## IMS XML-DB Interface Future

- SQL is a questionable IMS/XML interface
  - Hierarchical DB
  - Hierarchical Data
  - Relational Query Language??
- SQL/XML
  - Still relational
- XPath / XQuery
  - Only query right now
  - Still under development



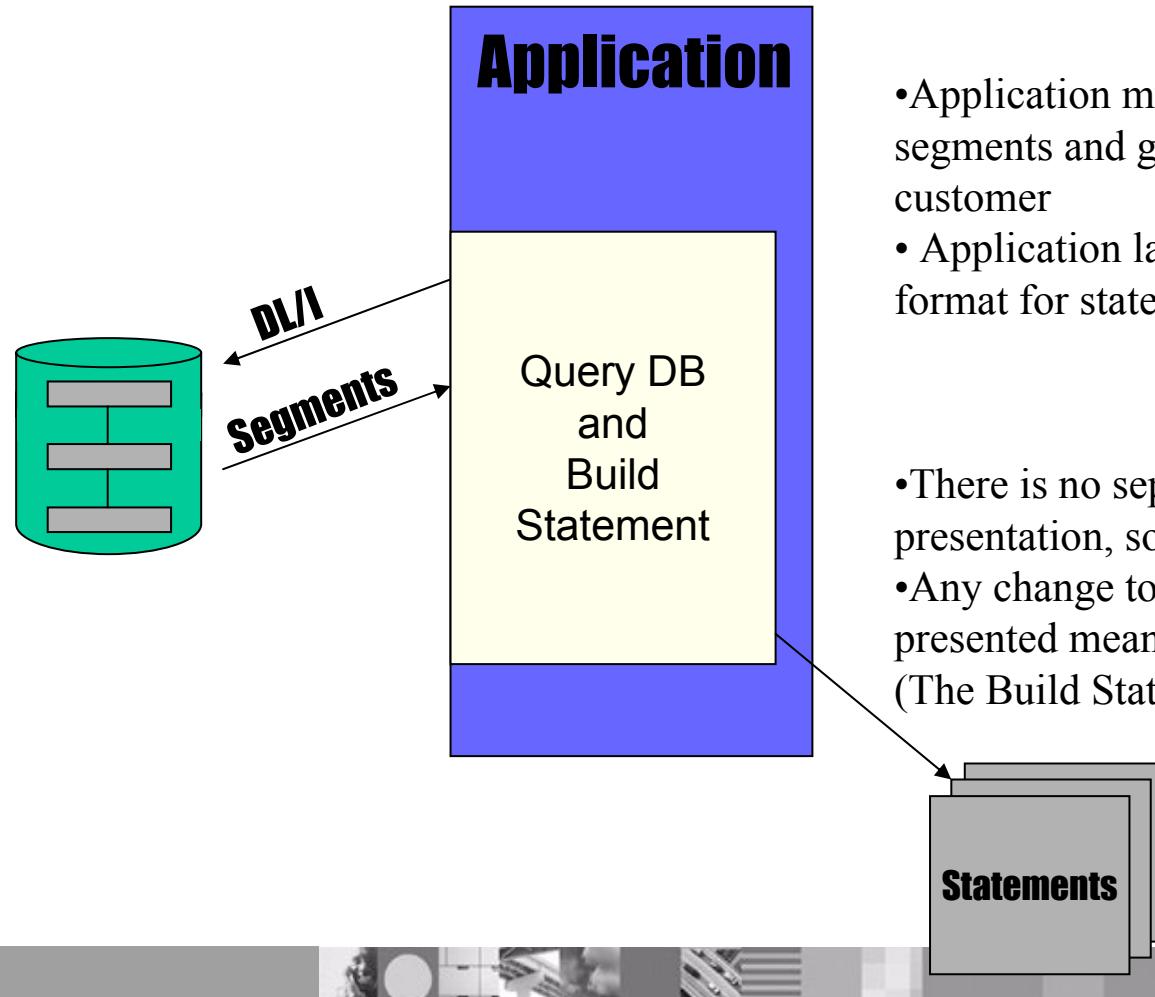
*the world depends on it*

## Hypothetical Bank DB



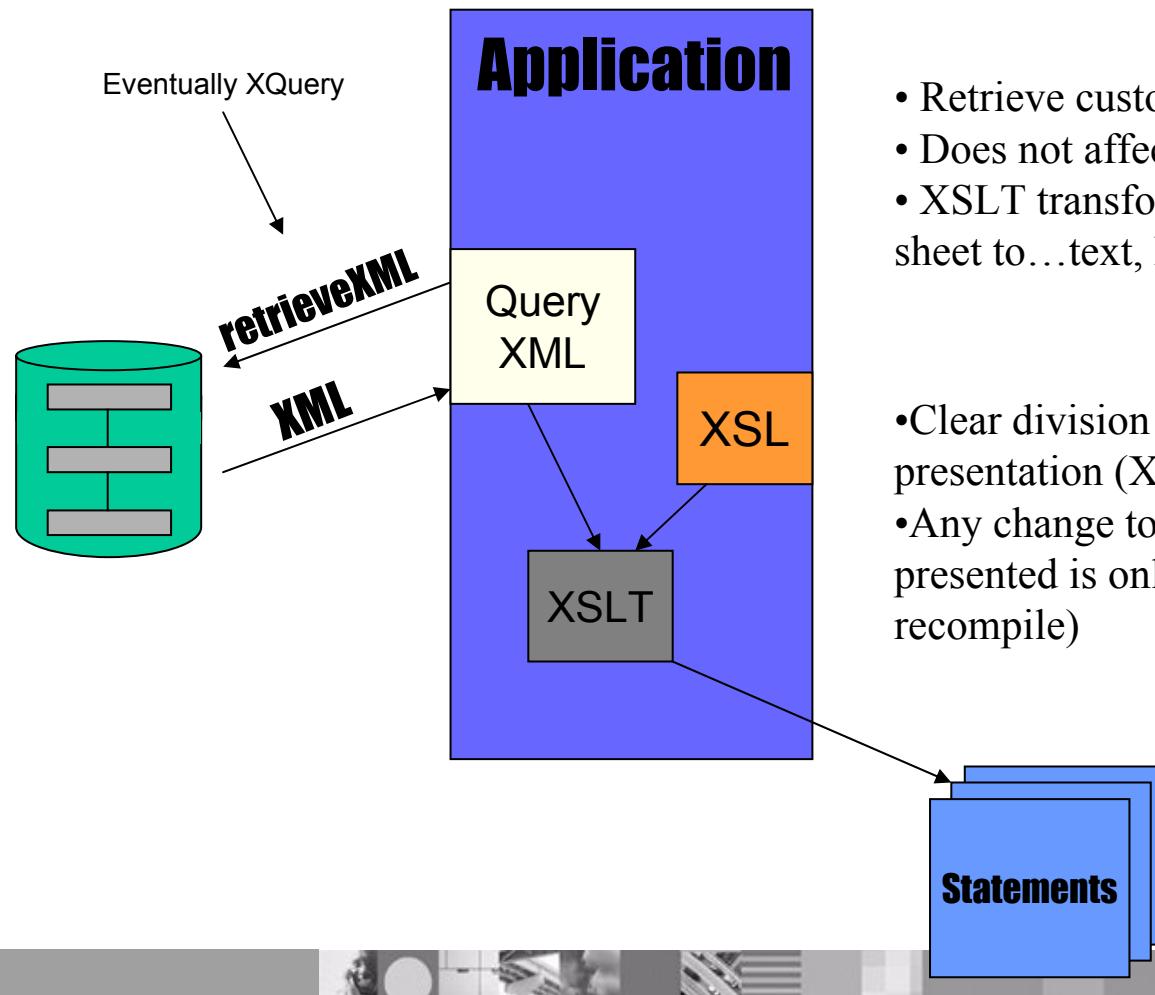
- Every month send customer statements
- On-line Account access
- etc.

## Current tedious design



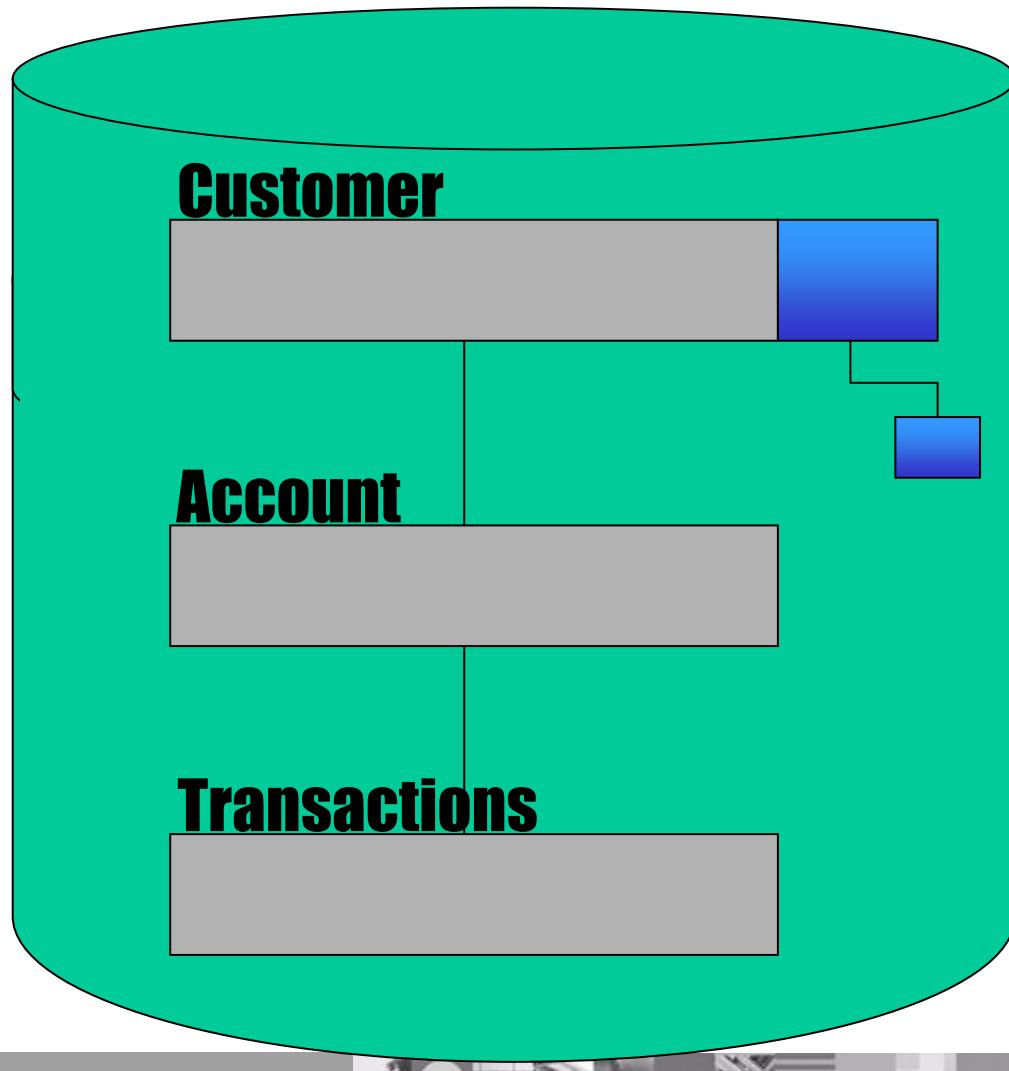
- Application must query all needed DL/I segments and gather needed data for each customer
- Application lays out data in desired output format for statement or web page.
- There is no separation of data and presentation, so
- Any change to the way the data is to be presented means a change to the application (The Build Statement Module).

## New XML design



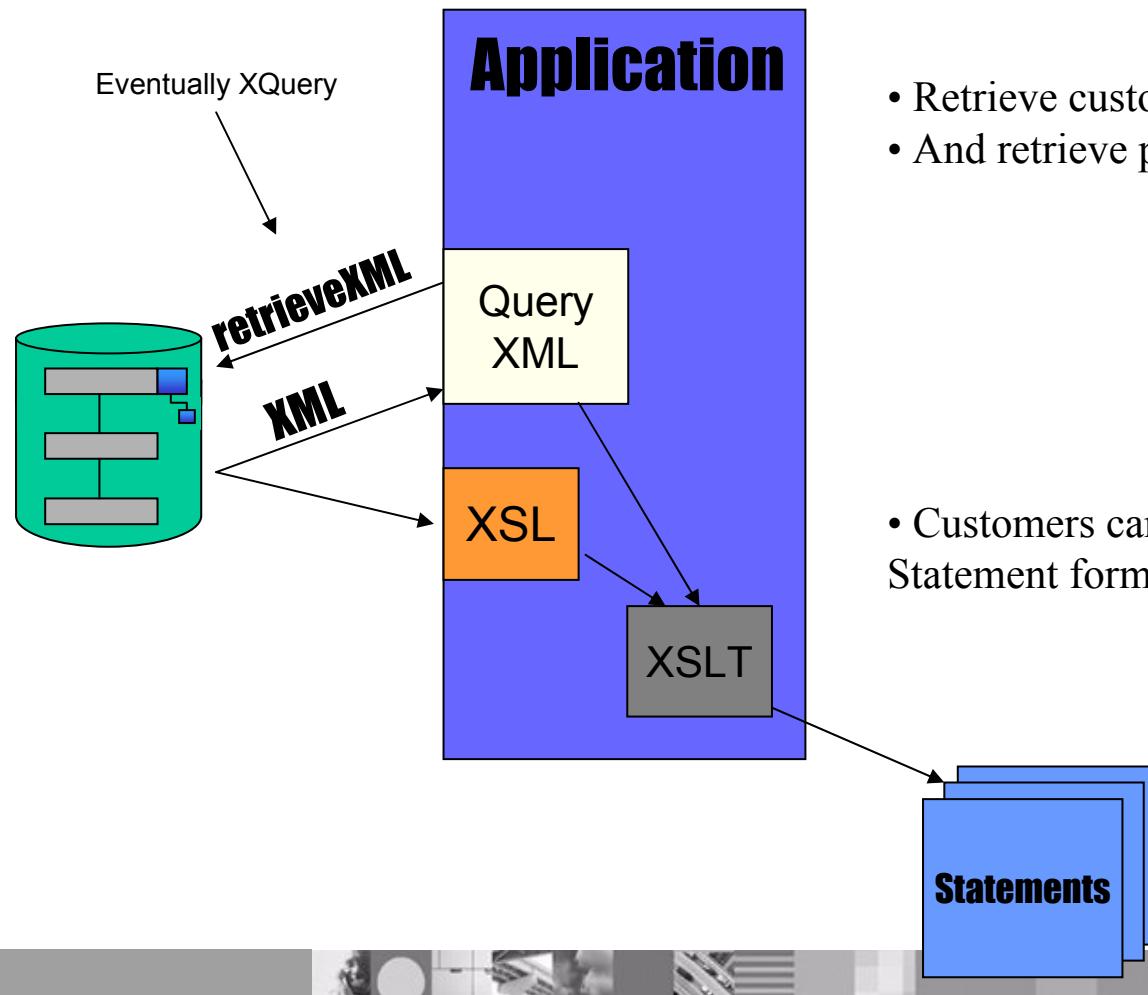
- Retrieve customer data in XML format
  - Does not affect other apps
  - XSLT transforms XML based on style sheet to...text, html, PDF, etc.
- 
- Clear division of data (XML from DB) and presentation (XSL)
  - Any change to the way the data is to be presented is only change to style sheet (no recompile)

## Extended DB with Intact Storage



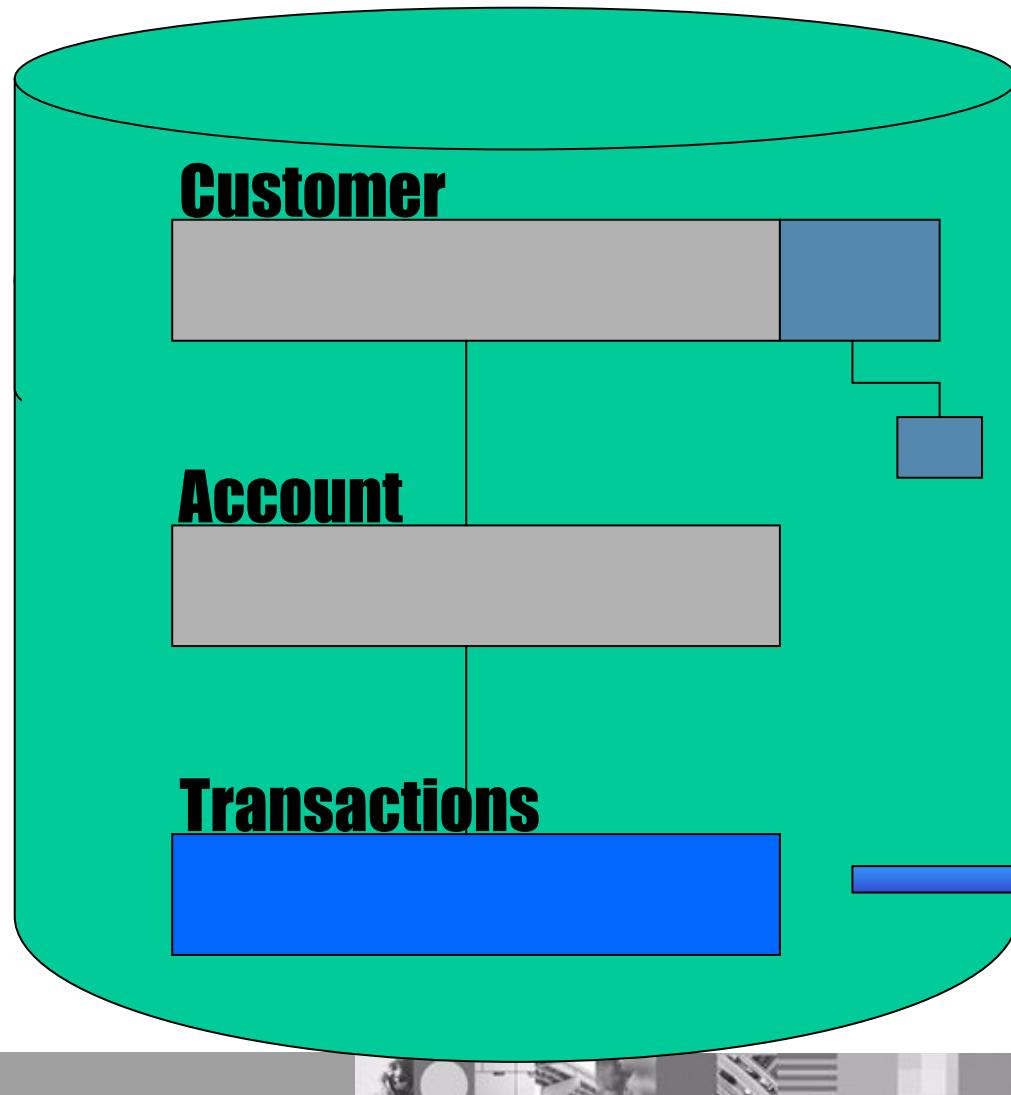
- Extend Customer with intact XML for each customers own personalized style sheet

## New XML design



- Retrieve customer data in XML format
  - And retrieve personalized XSL
- 
- Customers can change their own Bank Statement format

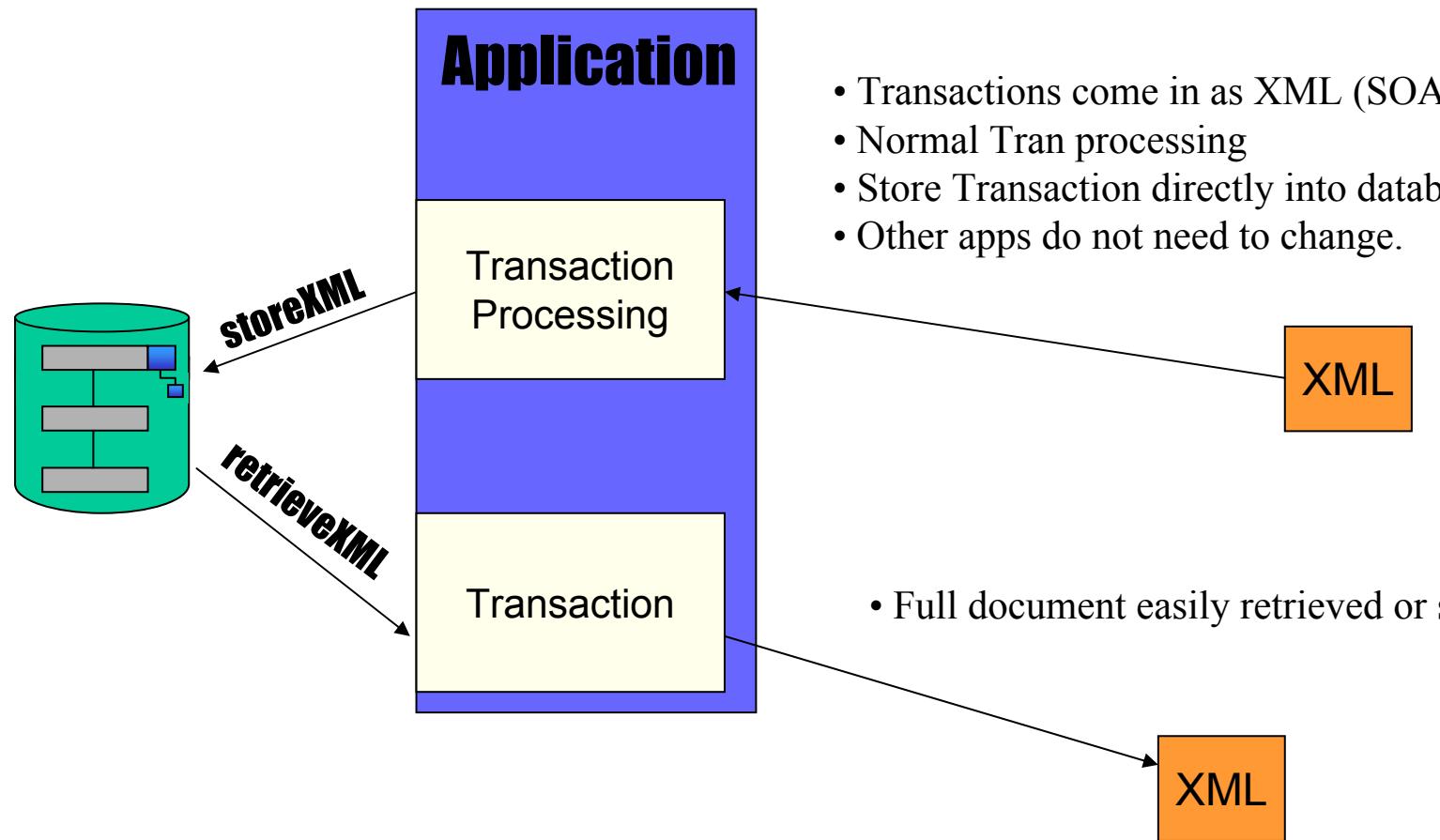
## Possible further extension



- Generate Schema for transactions and distribute to suppliers.

XML  
Schema

## New XML storage and retrieval



*the world depends on it*

# GUI DL/I Model Utility – Future

The screenshot shows the IMS - AutoDealer application interface. The main window displays a DL/I model diagram with several segments: MODEL, YEAR, MSRP, ORDER1, SALES, and STOCK. The ORDER1 segment is currently selected, with its fields ORDNBR, LASTNME, FIRSTNME, and TRUNCATEDNAMEEXAMPLE listed. The Properties panel at the bottom provides detailed information for the ORDNBR field, including:

- IMS Name: ORDNBR
- Alias: OrderNumber
- Data Type: CHAR
- Type Qualifier: (empty)
- Segment Length: 127 Bytes
- Start Byte: 1
- End Byte: 10

The interface also includes a Package Explorer showing project files like DLR\_PSB.java and DLR\_PSB\_Report.txt, and a Remote Systems view showing connections to QuatreB and ENGLAND. The top menu bar includes File, Edit, Navigate, Search, Project, Diagram, Modeling, Run, Window, and Help.