Data Normalization Reconsidered

Susan Malaika, Matthias Nicola
malaika@us.ibm.com, mnicola@us.ibm.com
10 October 2012

This presentation reflects the opinions of the authors
The authors welcome your feedback
Agenda

• Talk Description and Purpose
• Relational Databases
• Data Normalization
• Experiences with XML
• Recent Trends
• A Final Message
• References and Related Materials
• Backup
Talk Description

• Relational databases have been fundamental to business systems for more than 25 years. Data normalization is a methodology that minimizes data duplication to safeguard databases against logical and structural problems, such as data anomalies. Relational database normalization continues to be taught in universities and practiced widely. Normalization was devised in the 1970s when the assumptions about computer systems were different from what they are today.

• This talk provides a review of record keeping inside and outside of computer systems. Based on this background, the talk examines the problems associated with data normalization, such as complexity and the difficulty of mapping business records to normalized data in a changing world. The talk describes how the World Wide Web has impacted the creation and exchange of non-normalized business records. Alternative data representations, such as XML, JSON, and RDF to overcome normalization issues or to introduce schema flexibility, are covered.

• The talk is based on these articles on developerWorks:
  – Data Normalization Reconsidered - Part 1:
  – Data Normalization Reconsidered - Part 2:
Purpose of the Presentation

For the Future: To encourage further work on the way data is modeled and stored in computer systems.

For Today: To encourage data modelers to consider the storage models for their systems more carefully.

To Provide a Way of Thinking: To highlight the importance of utilizing data structures that represent real world business records in computer systems and that correspond to typical access patterns.
Relational Databases
## Knowledge Representation and Processing Examples

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>3800 BCE</td>
<td>Oldest known map appears in Mesopotamia.</td>
</tr>
<tr>
<td>3500 BCE</td>
<td>Cuneiform alphabet of ideogram pictures is created.</td>
</tr>
<tr>
<td>2800 BCE</td>
<td>Sumerian clay tablets record oil sales by amphorae.</td>
</tr>
<tr>
<td>1950 BCE</td>
<td>Hammurabi’s code of laws created such as &quot;eye for an eye&quot;.</td>
</tr>
<tr>
<td>700 BCE</td>
<td>Earliest known musical notation appears in India.</td>
</tr>
<tr>
<td>100 BCE</td>
<td>Paper first used in China.</td>
</tr>
<tr>
<td>1200s CE</td>
<td>Tally sticks used in China, split tally sticks used in medieval Europe</td>
</tr>
<tr>
<td>1940s CE</td>
<td>Sequential text files on punched cards for computer system <strong>input, output and storage</strong></td>
</tr>
<tr>
<td>1950s CE</td>
<td>Sequential text files on magnetic tapes for computer system <strong>input, output and storage</strong></td>
</tr>
<tr>
<td>1950s CE</td>
<td>Sequential text files on disk for computer system <strong>input, output and storage</strong>.</td>
</tr>
<tr>
<td>1960s CE</td>
<td>Databases for <strong>storage</strong> in computer systems.</td>
</tr>
<tr>
<td>1990s CE</td>
<td>JPEG format for computer system <strong>input, output and storage</strong>.</td>
</tr>
</tbody>
</table>

**Note:** The concept of databases introduced in the 1960s have the characteristic that you store something different from what you input or output from computer systems.
Some 1950s-1960s activities

Data Entry Staff

Moving a 5 Megabyte Disk in 1956

http://www.english.ucsb.edu/faculty/ayliu/unlocked/misc-images/punchcard_operators.jpg
http://sybarite.us/puertorico/2007/05/
The Computer of 1970

IBM SYSTEM/370 MODEL 145

Introduced in 1970, the Model 145 was the first IBM computer to have a main memory made entirely on monolithic circuits on silicon chips (previous 370 models used magnetic core main memories). Its system storage ranged from 112K to 512K, and it operated at speeds up to 11 times faster than the System/360 Model 30. Purchase prices ranged from about $705,775 to $1.78 million.

Note: 1.78 million dollars in 1970 for a computer with 512K memory.
Relational Databases

- Databases were introduced in the 1960s:
  - To make effective use of the new direct access storage devices
  - To provide multi-user access to shared data
- Two “data models” for databases arose in the 1960s that used addresses or pointers for application navigation:
  - The network model
  - The hierarchical model
- The relational model for databases was proposed by E.F. Codd in IBM in 1970 to:
  - Protect applications from knowing how stored data is organized (internally inside the computer)
  - Ensure all relationships are represented through values – avoiding pointer and link navigation
- The relational model introduced:
  - A universal data sub-language to be applied on relations
    - SQL - was the language that was implemented
  - A separation of data and meta-data
    - Stored data cannot be accessed unless through metadata
    - The universal query language uses the separate meta-data extensively to access the data that is stored

Data in the 1970s – Reality is Paper

Note: The relational model does not match external world business records nor does it match the inputs and outputs to applications.

People transform Business Records into input data for computers.

The External World

The Internal World of the Computer

Copyright IBM
Data Normalization
What is Data Normalization?  
As taught in the 1970s

- Normalization is the process of organizing data in a relational database into tables – with the goal of storing each piece of data, e.g., customer’s address, employees manager name, exactly once
  - To save disk space
  - To make updates in one place only when something in a business entity changes
- There are many normal forms – the goal is usually to reach the highest

### Purchase Order Snippets in a Non-Normalized Table

<table>
<thead>
<tr>
<th>Customer Name</th>
<th>Customer Country</th>
<th>Product Code</th>
<th>Product Name</th>
<th>Purchase Order Number</th>
<th>Date of Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gianfranco</td>
<td>Italy</td>
<td>333</td>
<td>Chair</td>
<td>92</td>
<td>21 June 2008</td>
</tr>
<tr>
<td>Jian</td>
<td>China</td>
<td>333</td>
<td>Chair</td>
<td>76</td>
<td>21 July 2007</td>
</tr>
<tr>
<td>Jian</td>
<td>China</td>
<td>555</td>
<td>Table</td>
<td>76</td>
<td>21 July 2007</td>
</tr>
</tbody>
</table>
### Customer Table

<table>
<thead>
<tr>
<th>ID</th>
<th>Customer Name</th>
<th>Customer Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gianfranco</td>
<td>Italy</td>
</tr>
<tr>
<td>2</td>
<td>Jian</td>
<td>China</td>
</tr>
</tbody>
</table>

### Product Table

<table>
<thead>
<tr>
<th>Product Code</th>
<th>Product Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>333</td>
<td>Chair</td>
</tr>
<tr>
<td>555</td>
<td>Table</td>
</tr>
</tbody>
</table>

### Purchase Order Table

<table>
<thead>
<tr>
<th>Purchase Order Number</th>
<th>Date of Order</th>
<th>Customer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td>21 June 2008</td>
<td>1</td>
</tr>
<tr>
<td>76</td>
<td>21 July 2007</td>
<td>2</td>
</tr>
</tbody>
</table>

### Purchase Product Table

<table>
<thead>
<tr>
<th>Purchase Order Number</th>
<th>Product Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td>333</td>
</tr>
<tr>
<td>76</td>
<td>333</td>
</tr>
<tr>
<td>76</td>
<td>555</td>
</tr>
</tbody>
</table>

Note: In this example only the latest values are shown – For example, Jian might have been living in the UK at the time of purchase order 76. So reconstructing the purchase order may be difficult because Jian’s address at that date is not available.
Data Normalization and Its Side Effects

Normalize to avoid data redundancy

--- redundant data wastes disk space
--- updates are made once

• **A note on redundant data:** Avoiding redundant data through normalization implies
  – Many intermediate “entities” exist inside the computer system defined by data designers
  – Business records have to be broken down into rows that are inserted into many tables
  – Joins are needed to recreate business records

• **A note on versioning:** The relational model does not embody an obvious context or versioning mechanism
  – After data items are modified, it is not always clear if:
    • new versions of the business entity have been created (customer moved to a new address) or
    • an error in the content was fixed (customer address was wrong)
  – Data designers are needed to interpret database content and to ensure that an audit log is maintained

• **A note on governance:** Normalization assumes updates – one of its main goal is to reduce update anomalies
  – However much business data must not be modified – (Accountants do not use erasers)
    • Append transactions to a bank account
    • Append a purchase order to a purchase order log
  – Some data is summary/roll-up – which is updated
    • Account balance

As disk space is more available, increasingly with compliance and governance strategies in place, more data has to remain unmodified. For example time-stamping mechanisms are introduced.
## Summarizing Normalization Effects

<table>
<thead>
<tr>
<th>The Non-Normalized External World</th>
<th>The Normalized Internal World</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business records exist as collections of facts that are true at a particular time</td>
<td>Business records are represented by many intermediate records that programs have to assemble and disassemble</td>
</tr>
<tr>
<td>Many versions of business records exist – required for audit and governance</td>
<td>Latest versions of intermediate records stored (to save disk space). Special programming needed to handle versioning</td>
</tr>
<tr>
<td>Business record structures change</td>
<td>Database re-design required when business record structures change</td>
</tr>
<tr>
<td>Business record structures are agreed across institutions. Many are familiar with the structures.</td>
<td>Custom normalized storage models exist for each institution that require specialist knowledge</td>
</tr>
</tbody>
</table>
Normalization Benefits & Hazards
As taught in 2006

• Benefits
  – Physical Space needed to store data is reduced
  – Avoid update anomalies
  – Normalizations allows changes to small amounts of data

• Hazards
  – Physical space not nearly as big a concern
  – Too many tables lead to too many joins
  – People don’t understand the model (only the data architects do)
  – Custom models – every company does its own

Derived from “Beginning Database Design”, Gavin Powell, Wiley 2006,
Pressures on Normalization

• De-normalize for performance
  – Store de-normalized data to avoid joins, e.g., e-Bay, Flickr, Google Big Table
• De-normalize for data warehousing
  – Create de-normalized star schemas (schemas used in data warehouses that are easier for users to understand)
• De-normalize for governance
  – Store an audit log of business records
• De-normalize for improved understanding
  – No need to review complex designs and metadata to understand what’s stored

Note: So why normalize data for storage?
Industry Standards Trends

- Companies are using the industry exchange formats inside their systems, and with other companies e.g., FpML in financial markets
- Increasingly the industry exchange formats are becoming the points of compliance, e.g., SEPA – Single European Payments area (banking)
- Governments are mandating the use of the formats, e.g., HL7 CDA (healthcare)

Note: Skills in the industry exchange formats are increasing – and storage models based around the exchange formats are becoming well understood
Financial Application with FpML
(Financial Products Markup Language)
Used to describe Over the Counter Derivatives Products

Note: Just a part of the FpML schema requires over 400 tables
Reality is Digital: Data in the 21\textsuperscript{st} Century

Note: The relational storage model does not match external world business records. Although, business records do match the inputs and outputs to applications.

**Business Records:**
- Invoices - XML
- Forms - XML
- Orders - XML
- Statements - XML

**Structured to support the needs of the business or the users**

**People no longer transform business records on paper into input data for computers**

The External World

**Input / Output**
- XML, JSON etc
- XML, JSON, etc

**Data at the Edges**

The Internal World of the Computer

**Relational Tables**

**App (XML, JSON etc)**

**App (objects)**

Database experts are needed to explain what is stored
Experiences with XML Database
Normalized vs. Non-Normalized Data

- Non-normalized
- Ready for use
- Easy to store & retrieve
- Easy to understand

- Normalized
- Not ready for use
- Storage & retrieval requires (dis)assembly,
- Hard to understand except for engineers!
Normalized vs. Non-Normalized Data

XML Business Record

Application Server

DB2

- Non-normalized
- Ready for use
- Easy to store & retrieve
- Easy to understand

Normalized vs. Non-Normalized Data

XML Business Record

Application Server

DB2

- Normalized
- Not ready for use
- Storage & retrieval requires (dis)assembly,
- Hard to understand except for engineers!

Copyright IBM
Normalized vs. Non-Normalized Data

- Non-normalized
- Ready for use
- Easy to store & retrieve
- Easy to understand

- Normalized
- Not ready for use
- Storage & retrieval requires (dis)assembly,
- Hard to understand except for engineers!

Copyright IBM
## Normalized

**Customer**

<table>
<thead>
<tr>
<th>Cid</th>
<th>FirstName</th>
<th>LastName</th>
<th>DateOfBirth</th>
<th>Nationality</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>John</td>
<td>Doe</td>
<td>1965-09-27</td>
<td>German</td>
</tr>
</tbody>
</table>

**Account**

<table>
<thead>
<tr>
<th>Cid</th>
<th>AccNo</th>
<th>Currency</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>985739476</td>
<td>Euro</td>
<td>120,000</td>
</tr>
<tr>
<td>12345</td>
<td>985710938</td>
<td>Euro</td>
<td>2786.23</td>
</tr>
<tr>
<td>12345</td>
<td>985808142</td>
<td>USD</td>
<td>523,891</td>
</tr>
</tbody>
</table>

**Positions**

<table>
<thead>
<tr>
<th>AccNo</th>
<th>Symbol</th>
<th>Shares</th>
</tr>
</thead>
<tbody>
<tr>
<td>985739476</td>
<td>IBM</td>
<td>1,200</td>
</tr>
<tr>
<td>985739476</td>
<td>ORCL</td>
<td>2,500</td>
</tr>
<tr>
<td>985739476</td>
<td>VFINX</td>
<td>550</td>
</tr>
<tr>
<td>985710938</td>
<td>SBTYA</td>
<td>12,000</td>
</tr>
<tr>
<td>985710938</td>
<td>VIVAX</td>
<td>75</td>
</tr>
<tr>
<td>985808142</td>
<td>DWCT</td>
<td>1,128</td>
</tr>
<tr>
<td>985808142</td>
<td>PMCOA</td>
<td>8,461</td>
</tr>
</tbody>
</table>

### Multi-way join and/or multiple SQL stmts necessary to get all data for 1 customer.

## Non-normalized

**CustomerAccounts**

```xml
<Customer Cid="12345">
  <FirstName>John</FirstName>
  <LastName>Doe</LastName>
  <DateOfBirth>1965-09-27</DateOfBirth>
  <Nationality>German</Nationality>
  <Accounts>
    <Account AccNo="985739476">
      <Currency>Euro</Currency>
      <Balance>120,000</Balance>
      <Positions>
        <Stock Sym="IBM" Shares="1,200"/>
        ...
      </Positions>
    </Account>
    <Account AccNo="985710938">
      <Currency>Euro</Currency>
      <Balance>2786.23</Balance>
      <Positions>
        ...
      </Positions>
    </Account>
    <Account AccNo="985808142">
      <Currency>USD</Currency>
      ...
    </Account>
  </Accounts>
</Customer>
```

### Hierarchical structure avoids data redundancy!

Get entire business record in a single fetch.
Normalized XML vs. Non-normalized Relational Data

Performance of retrieving business objects from:

(a) 12 normalized relational tables, using 12 SQL queries
(b) single XML column, using 1 SQL/XML query

Relational: capacity of the test machine exhausted at 84 concurrent users.
Summary: XML vs. Relational Performance

• XML can be faster than Relational
  – for object-centric access patterns, i.e.:
  – if applications need to store & retrieve entire business objects that would otherwise be spread across multiple tables

• Relational can be faster than XML
  – for column-oriented access patterns, i.e. queries that read only a small piece of data from each of many objects
  – aggregation queries, summaries, warehousing
  – e.g. sum of all account balances

Know your workload!
XML and relational are not mutually exclusive! Consider a hybrid XML/relational database schema

- Combine the benefits of both relational and XML

<table>
<thead>
<tr>
<th>Relational columns</th>
<th>XML columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good for flat, structured data</td>
<td>Good for hierarchical, nested data</td>
</tr>
<tr>
<td>Good for fixed schemas</td>
<td>Good for schema flexibility</td>
</tr>
<tr>
<td>Good if all records have the same format</td>
<td>Good if record format can vary from row to row, sparse data</td>
</tr>
<tr>
<td>Good for strongly typed data</td>
<td>Good for strongly typed, somewhat typed, or un-typed data</td>
</tr>
<tr>
<td>Good for partitioning keys, clustering keys,</td>
<td>Good if the data format outside the database is also XML (app, ESB, etc.)</td>
</tr>
<tr>
<td>multi-column indexes, etc.</td>
<td></td>
</tr>
<tr>
<td>Can be faster than XML</td>
<td>Can be faster than Relational</td>
</tr>
</tbody>
</table>

Copyright IBM
Recent Trends
Two Significant Trends
– Both driven by the Web
- Both enabling new applications of data

3rd Normal Form
Variant on row based stores is column based stores

(1) De-normalized or Not-normalized
LOBs, XML, JSON, Documents etc

(2) Highly Normalized
RDF (Resource Definition Framework) Triples and Ontologies

See “Data Normalization Reconsidered” –
# Data Characteristics and Guidelines

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Intact Data (Not Normalized)</th>
<th>RDF (Highly Normalized)</th>
</tr>
</thead>
</table>
| **Identifiers** | • Identifiers are usually values, e.g., SSN, ISBN - global identifiers such as URLs are usually generated via REST / Web APIs  
• Schemas can be globally or locally defined  
• Query, Transformation & Schema Languages exist or emerging | • Global Identifiers are used throughout to facilitate integration: URIs; Linked Data – URLs  
• Ontologies are typically globally defined  
• Query, Transformation & Schema Languages exist or emerging |
| **Usage Guideline** | Use intact data when it:  
• matches the typical unit of retrieval and manipulation, e.g., data exchange, audit and logging use cases  
• is the unit of integrity and versioning, e.g., a business record | Use RDF when it:  
• matches typical unit of retrieval and manipulation, e.g., integration and inferencing use cases |
| **Note:** | | Note:  
• RDF is usually unsuitable for managing records that need coordinated integrity or to be versioned.  
• RDF usually represents the latest version only |
Hybrid Data

**XML – Complete business records**
- Good for representing business records that are shared, for schema flexibility, for versioning
- Query language: XPath and XQuery
- Proposals to incorporate JSON support into XQuery at the W3C

**Relational – Third normal form**
- Versatile, works for many scenarios,
- Typically normalized to 3rd normal form
- Avoid update anomalies and save storage
- Sometimes then de-normalized for improved understanding or performance
- Query language: SQL

**RDF (Resource Description Framework) triples, Linked Data, Graph Data**
- Good for data about things, for sharing data definitions, for relationships, for inferencing, for schema flexibility
- Part of the movement from the Web of Documents to the Web of Data
- Highly normalized
- Query language: SPARQL

**Hybrid query and manipulation languages:**
- Relational and XML: Standardized integration of XQuery and SQL (SQL/XML)
- RDF (triples): No hybrid language for integration with relational

Copyright IBM
# Relational, XML and RDF

<table>
<thead>
<tr>
<th>Relational</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables <img src="image" alt="Table" /></td>
<td>Trees <img src="image" alt="Tree" /></td>
<td>Graphs <img src="image" alt="Graph" /></td>
</tr>
<tr>
<td>Flat, highly structured</td>
<td>Hierarchical data</td>
<td>Linked data</td>
</tr>
<tr>
<td>Multiple rows in multiple tables represent a business record</td>
<td>Trees represent business records</td>
<td>Triples represent business records and their properties via URIs</td>
</tr>
<tr>
<td>Fixed schema</td>
<td>No or flexible schema</td>
<td>Highly flexible</td>
</tr>
<tr>
<td>SQL (ANSI/ISO)</td>
<td>XPath/XQuery (W3C)</td>
<td>SPARQL (W3C)</td>
</tr>
</tbody>
</table>
## Summary: Suitability of Normalized versus Non-normalized Storage

<table>
<thead>
<tr>
<th></th>
<th>Suitable for non-normalized data representation, for example, XML</th>
<th>Suitable for normalized or semi-denormalized data representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Data access is &quot;object-centric&quot; (all or most pieces of a business record are accessed together)</td>
<td>Data access is set-oriented or column-oriented, for example for analytics</td>
</tr>
<tr>
<td>2</td>
<td>Intact business records are exchanged via web services or SOA</td>
<td>Original business records do not need to be reassembled</td>
</tr>
<tr>
<td>3</td>
<td>Versioning is required: updates are replaced by inserts of immutable versions</td>
<td>Only the latest state of each business record needs to be retained</td>
</tr>
<tr>
<td>4</td>
<td>Schema evolution</td>
<td>Schema is mature, stable, unlikely to evolve</td>
</tr>
<tr>
<td>5</td>
<td>Auditing and compliance of business records are critical</td>
<td>Audit/compliance requirements are short-term, weak, or absent</td>
</tr>
</tbody>
</table>

Consider RDF for linking data across heterogeneous data sources, inferencing
A Final Message

The storage model should correspond to external business records and access patterns to produce:

- Easy to understand systems
- Agile development
- Applications that perform well
More Related Materials

- Punched Cards [http://en.wikipedia.org/wiki/Punch_card](http://en.wikipedia.org/wiki/Punch_card)
- Normalized Data is for Sissies [http://www.kottke.org/04/10/normalized-data](http://www.kottke.org/04/10/normalized-data)
Backup
# Some Data and Metadata Standards

<table>
<thead>
<tr>
<th></th>
<th>Relational</th>
<th>XML</th>
<th>JSON</th>
<th>Linked Data / RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Metadata</strong></td>
<td>Data Definition Language</td>
<td>XML Schema XSD (W3C),</td>
<td>JSON Schema - IETF</td>
<td>RDFS (RDF Schema), Ontology</td>
</tr>
<tr>
<td>(ISO)</td>
<td>(ISO)</td>
<td>Namespaces (W3C)</td>
<td></td>
<td>(W3C and elsewhere)</td>
</tr>
<tr>
<td><strong>Constraints</strong></td>
<td>Integrity Constraints in</td>
<td>Schematron (ISO), Relax-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>table definitions (ISO)</td>
<td>table definitions (ISO)</td>
<td>NG (OASIS)</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td><strong>Triggers</strong></td>
<td>Relational triggers</td>
<td>-</td>
<td>-</td>
<td>RIF (W3C)</td>
</tr>
<tr>
<td>**Data Exchange</td>
<td>SQL standard (ISO) defines</td>
<td>XML is a syntax widely used</td>
<td>JSON is a serialized format,</td>
<td>XML, Turtle</td>
</tr>
<tr>
<td>Serializations</td>
<td>an XML serialization but it</td>
<td>for data exchange (W3C)</td>
<td>there are XML representations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>is not widely used – There</td>
<td></td>
<td>of JSON too</td>
<td></td>
</tr>
<tr>
<td></td>
<td>is no agreed JSON serialization.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>There are RDF serializations.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Annotations</strong></td>
<td>Not part of the relational</td>
<td>Many kinds of annotations</td>
<td>-</td>
<td>RDFa (W3C) can be used to annotate XML</td>
</tr>
<tr>
<td>model</td>
<td>model</td>
<td>are defined for XML schemas</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>and for XML data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>**Query &amp; CRUD</td>
<td>Data Manipulation Language</td>
<td>XPath, XQuery (W3C) and</td>
<td>JAQL, JSONiq, database</td>
<td>SPARQL (W3C)</td>
</tr>
<tr>
<td>Languages</td>
<td>(DML), SQL, SQL/XML (ISO)</td>
<td>others for CRUD</td>
<td>specific languages</td>
<td></td>
</tr>
<tr>
<td>**Query &amp; CRUD</td>
<td>Many for various programming</td>
<td>Various, includes some of</td>
<td>-</td>
<td>SPARQL Graph Store HTTP Protocol</td>
</tr>
<tr>
<td>APIs</td>
<td>languages, e.g., JDBC, ODBC</td>
<td>the relational APIs and</td>
<td></td>
<td>– for CRUD (W3C)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specific APIs, e.g., XQJ</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Collection</strong></td>
<td>Table, View, Database (ISO)</td>
<td>XML Collection Function</td>
<td>-</td>
<td>RDF Graphs (W3C)</td>
</tr>
<tr>
<td></td>
<td>(ISO)</td>
<td>(W3C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>**Transformation &amp;</td>
<td>SQL (Tables to Tables)</td>
<td>XSLT (XML to text, includes</td>
<td>JavaScript</td>
<td>-</td>
</tr>
<tr>
<td>Other Languages</td>
<td></td>
<td>XML); XForms</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL XMLTABLE (XML to</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>relational)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some XML communities who work on query and transformation languages are reviewing the idea of XML languages supporting JSON.

Currently, the role of JSON is for data exchange between JavaScript clients and servers, and it has emerged as a data storage format.
# XML Schema Characteristics for Some Industry Formats [1]

<table>
<thead>
<tr>
<th>Industry Format</th>
<th>Version</th>
<th>Types</th>
<th>Elements</th>
<th>Attributes</th>
<th>XSD's</th>
<th>Max. XSD size in kB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACORD (Association for Cooperative Operations Research and Development)</td>
<td>(XMLife)</td>
<td>1369</td>
<td>9378</td>
<td>1275</td>
<td>4</td>
<td>743</td>
</tr>
<tr>
<td>ARTS (Association for Retail Technology Standards)</td>
<td>1.0 - 3.0</td>
<td>4825</td>
<td>6305</td>
<td>2011</td>
<td>32</td>
<td>625</td>
</tr>
<tr>
<td>CDISC (Clinical Data Interchange Standards Consortium)</td>
<td>00-9-03</td>
<td>98</td>
<td>84</td>
<td>71</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>FpML (Financial Products Markup Language)</td>
<td>4.2</td>
<td>686</td>
<td>1867</td>
<td>196</td>
<td>23</td>
<td>130</td>
</tr>
<tr>
<td>FIXML (Financial Information Exchange)</td>
<td>4.4</td>
<td>1310</td>
<td>619</td>
<td>2593</td>
<td>41</td>
<td>797</td>
</tr>
<tr>
<td>HL7CDA (Clinical Document Architecture)</td>
<td>3</td>
<td>1953</td>
<td>945</td>
<td>477</td>
<td>6</td>
<td>749</td>
</tr>
<tr>
<td>IRS1120 (IRS e-File Form 1120)</td>
<td>2006v3.3</td>
<td>3415</td>
<td>11591</td>
<td>2632</td>
<td>600</td>
<td>214</td>
</tr>
<tr>
<td>MISMO (Mortgage Industry Standards Maintenance Organization)</td>
<td>2.3 - 2.4</td>
<td>2899</td>
<td>1087</td>
<td>13733</td>
<td>31</td>
<td>2865</td>
</tr>
</tbody>
</table>

Copyright IBM

<table>
<thead>
<tr>
<th>Industry Format</th>
<th>Version</th>
<th>Types</th>
<th>Elements</th>
<th>Attributes</th>
<th>XSD's</th>
<th>Max. XSD size in kB</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCJE (NIEM) (Minnesota Criminal Justice Event (National Information Exchange Model))</td>
<td>1</td>
<td>415</td>
<td>936</td>
<td>46</td>
<td>7</td>
<td>1661</td>
</tr>
<tr>
<td>OTA (OpenTravel Alliance)</td>
<td>2003/5</td>
<td>27293</td>
<td>24893</td>
<td>43141</td>
<td>234</td>
<td>538</td>
</tr>
<tr>
<td>STAR (Standards for Technology in Automotive Retail – OAGIS)</td>
<td>5.0.4</td>
<td>5846</td>
<td>77319</td>
<td>625</td>
<td>192</td>
<td>1200</td>
</tr>
<tr>
<td>TWIST (Transaction Workflow Innovation Standards Team)</td>
<td>3.1</td>
<td>1016</td>
<td>2314</td>
<td>20</td>
<td>19</td>
<td>154</td>
</tr>
<tr>
<td>UBL (Universal Business Language)</td>
<td>2</td>
<td>682</td>
<td>2665</td>
<td>253</td>
<td>43</td>
<td>938</td>
</tr>
<tr>
<td>UNIFI (ISO20022 UNiversal Financial Industry message scheme)</td>
<td>1.01 - 2.01</td>
<td>5082</td>
<td>9747</td>
<td>127</td>
<td>71</td>
<td>42</td>
</tr>
<tr>
<td>XBRL - GL (Extensible Business Reporting Language - Global Ledger)</td>
<td>2006-10-25</td>
<td>1858</td>
<td>2847</td>
<td>383</td>
<td>45</td>
<td>39</td>
</tr>
</tbody>
</table>

*These tables are intended to give an idea of the different design styles for schemas – The schemas may have moved on since these tables was created.*