Data-driven systems

Data dictionaries
Semi-structured data (XML, JSON)

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Contents

- Data dictionaries
- Handling semi-structured data
  - XML, JSON
  - XML handling in relational data bases

Data dictionaries of relational databases

What is the data dictionary used for?

```
IF  EXISTS (SELECT * FROM sys.objects
       WHERE object_id = OBJECT_ID(N'[dbo].Invoice’)
              AND type in (N'U’))
  drop table Invoice

create table [Invoice](...)
```

What is the data dictionary used for?

```
IF  EXISTS (SELECT * FROM sys.objects
       WHERE object_id = OBJECT_ID(N'[dbo].Invoice’)
              AND type in (N'U’))
  drop table Invoice

create table [Invoice](...)
```

- Create script of our sample database
- Why?
  - Idempotent script: yields the same result regardless of the current state of the database

Data dictionary

- Data dictionary
- Central meta-store describing the stored data, its format, relationships, etc.
  - E.g. name of the tables, name of columns, data types
- (Can also be a document.)
- Integral part of the database
  - Read-only views.
  - Can be used on DML and DLL statements too.
    - E.g.: schema migration: create the table if it does not yet exist

Contents of the data dictionary

- Description of every object in the database
  - Tables, views, indices, sequences, stored procedures, …
- Integrity requirements
- Users, permissions
- Monitoring information
  - E.g. number of active connections to the server
- Auditing information
  - E.g. how edited schema objects

MS SQL Server data dictionary

- Information Schema Views
  - According to ISO standard
  - `INFORMATION_SCHEMA.`
    - `TABLES, VIEWS, COLUMNS, PARAMETERS, TABLE_PRIVILEGES, …`
- Catalog Views
  - Information about the status of the server.
  - `sys.`
    - `databases, database_files, filegroups, messages, schemas, objects, tables, columns, foreign_keys,`
- Dynamic Management Views
  - Diagnostic information about the server.
  - `sys.dm_tran_locks, sys.dm_exec_cached_plans, sys.dm_exec_sessions`

MS SQL Server data dictionary example

```
select * from sys.tables
select * from INFORMATION_SCHEMA.TABLES
select * from sys.objects
select * from INFORMATION_SCHEMA.COLUMNS
```

schema	table	column	index	default	nullable	type
dbo	VAT	ID	1	NULL	NO	int
dbo	VAT	Percentag	2	NULL	YES	int
dbo	PaymentMethod	ID	1	NULL	NO	int
dbo	PaymentMethod	Mode	2	NULL	YES	nvarchar
dbo	PaymentMethod	Deadline	3	NULL	YES	nvarchar
dbo	Status	ID	1	NULL	NO	int

MS SQL Server data dictionary example

```
IF EXISTS (
      SELECT * FROM sys.objects
      WHERE type = 'U' AND name = 'Product')
DROP TABLE Product
```

```
if NOT EXISTS(
      SELECT * FROM INFORMATION_SCHEMA.COLUMNS
      WHERE TABLE_NAME = 'Product',
      AND COLUMN_NAME = 'Description')
alter table Product add Description xml;
```

Handling semi-structured data

XML

XML: Extensible Markup Language

- Textual representation of data in a platform independent manner.
- Easy to read for humans, parse by computers.
- Goal: simple, general usage.
- Originally: designed for document storage.
  - Used at a wide variety of places: RSS, Atom, SOAP, OpenXML, XHTML, OpenDocument, …
- Self-descriptive.

```
<course>
  <name>Data-driven systems</name>
  <code>VIAUAC01</code>
</course>
```

Content of XML documents:

```
<?xml version="1.0"   XML declaration
        encoding="UTF-8"?>
<!-- comment -->
<element attribute="value">
  <tag>content</tag>
  <![CDATA[ any content ]]>
</element>
```

XML - example

```
public class Customer
{
    public string Name;
    public DateTime Registered;
    public Address Address;
}
```

```
public class Address
{
    public string City;
    public int ZipCode;
}
```

```
<?xml version="1.0"?>
<Customer>
  <Name>John Doe</Name>
  <Registered>2016-10-26T08:58:26.6412829+02:00</Registered>
  <Address>
    <City>Budapest</City>
    <ZipCode>1118</ZipCode>
  </Address>
</Customer>
```

Encoding

- How to translate characters into bytes
- ASCII: 0-127 English characters -> 1 byte / character
- The code page indicates what character the codes above 127 represent
- For example, ISO/IEC 8859-1 does not include ‘ő', only 'õ'
- Unicode: every abstract character gets a number
- UTF-8: variable length encoding 1 or 4
- UTF-16: variable length encoding 2 or 4
- XML: the parser knows from the string '<?xml' whether it is 1 or 2 bytes, high endian, etc., but the code page is still required

Namespaces

- Just like namespaces in C++ and C#, and packages in Java
- Tag names can be arbitrary: clash.
- Namespace is a prefix: <**ns**:tag>
- Needs to be declared: *xmlns:ns="URI"*

```
<ns:tag xmlns:ns="http://www.aut.bme.hu">

<root xmlns:ns="http://www.aut.bme.hu">
   <ns:tag>abc<ns:tag>
</root>
```

- Default namespace

```
<root xmlns="http://www.aut.bme.hu">
```

Namespaces example: HTML and XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
/1999/XSL/Transform"><xsl:template match="/">
<html><body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr>
      <th style="text-align:left">Title</th>
      <th style="text-align:left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
</body></html>
</xsl:template></xsl:stylesheet>
```

XML - drawbacks

- Textual representation of data
  - Confined to a single platform -> no issues
  - Conforming to a standard (e.g. SOAP)
  - Document the schema, e.g. XSD
- Data types without specification
  - Date, bool
- Ambiguous data representation
  - Attribute? Child element? Null?
- Textual -> larger size

XML - .NET

- System.Xml.Serialization.XmlSerializer

```
var ser = new XmlSerializer(typeof(C));
ser.Serialize(<stream>, <obj>);
myobj = (C)ser.Deserialize(<stream>);
```

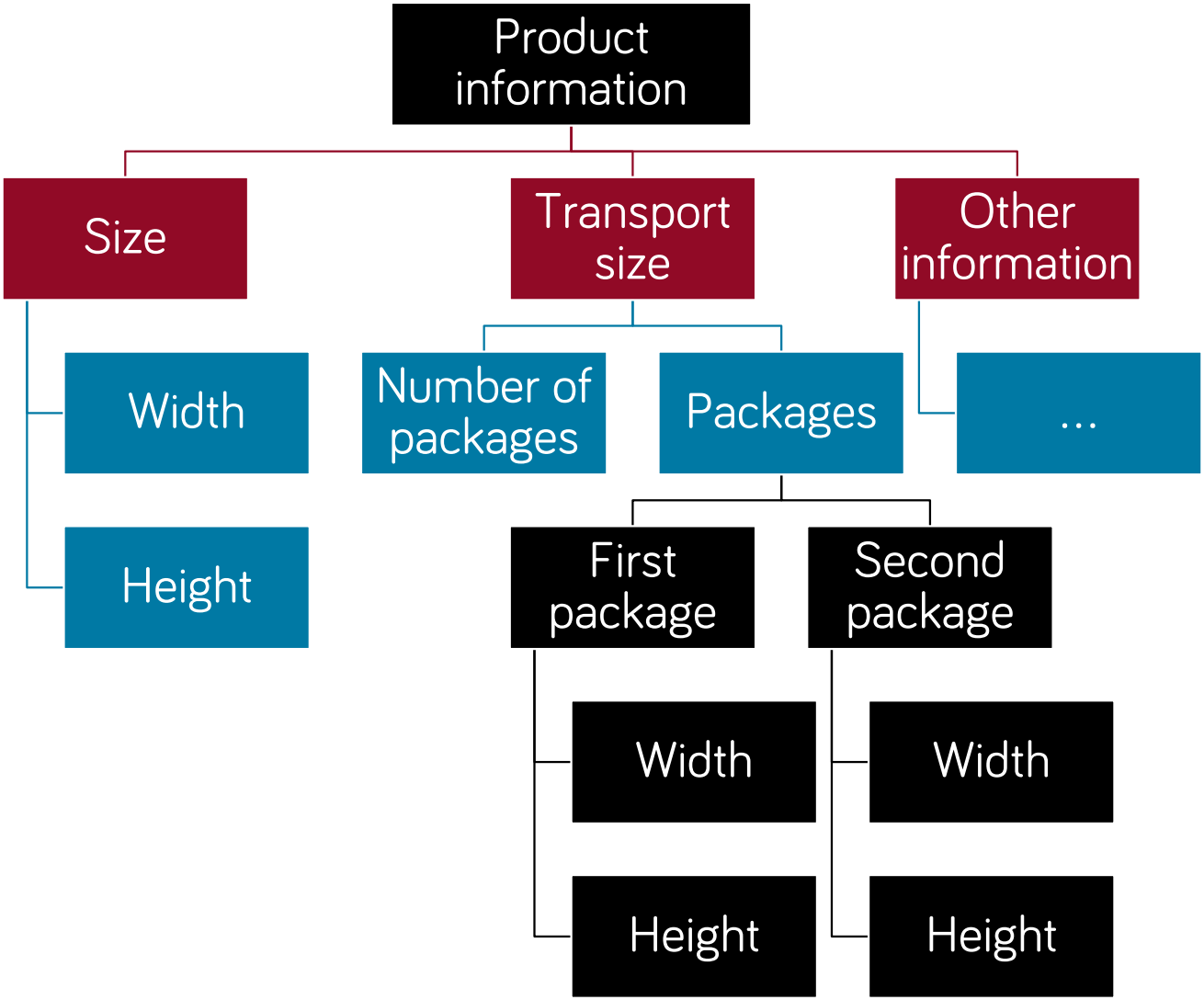
- Customization using C# attributes

```
[XmlElement("addr")]
public class Address
{
    [XmlAttribute("cty")]
    public string City;
    [XmlIgnore]
    public int CalculatedDistance;
}
```

Schema

- Xml document is **well-formed**, if
  - The content syntax is correct.
    - every opening tag has a closing pair, with correct nesting
    - has a single root element
    - Etc.
- Content **valid**ity is a different question.
  - Are the tag names correct?
  - Are the content of tags as expected?
  - Can be described with a schema, such as DTD, XSD
- Validation: does the XML document correspond to the schema?
  - Can be evaluated programmatically.

DOM: Document Object Model



XPath

- Standard XML query language
- Queries a part of the XML document
- Returns a node or boolean/number/text data
- It is closely related to XSLT

XPath

```
<library>
 <title>My book collection</title>
 <book>
    <title lang="en">Harry Potter</title>
    <price>1234</price>
 </book>
 <book>
    <title lang="hu">Data-driven systems</title>
    <price>5678</price>
 </book>
</library>
```

XPath

```
<library>
 <title>My book collection</title>
 <book>
    <title lang="en">Harry Potter</title>
    <price>1234</price>
 </book>
 <book>
    <title lang="hu">Data-driven systems</title>
    <price>5678</price>
 </book>
</library>
```

library/book

/library/book

XPath

tagname	Name of an element with the specified name
/	From the root
//	From the current node in descendents
.	Actual node
..	Parent node
@name	Attribute with the specified name
/bookstore/book[1]	Child, indexed from 1
/bookstore/book[last()]	Last child
/bookstore/book[position()<3]	First two children
//title[@lang='en']	'title' element having attribute 'lang' with value 'en'

XPath

```
<library>
 <title>My book collection</title>
 <book>
    <title lang="en">Harry Potter</title>
    <price>1234</price>
 </book>
 <book>
    <title lang="hu">Data-driven systems</title>
    <price>5678</price>
 </book>
</library>
```

//book

XPath

```
<library>
 <title>My book collection</title>
  <book>
    <title lang="en">Harry Potter</title>
    <price>1234</price>
  </book>
  <book>
    <title lang="hu">Data-driven systems</title>
    <price>5678</price>
  </book>
</library>
```

//title

XPath

```
<library>
 <title>My book collection</title>
 <book>
    <title lang="en">Harry Potter</title>
    <price>1234</price>
 </book>
 <book>
    <title lang="hu">Data-driven systems</title>
    <price>5678</price>
 </book>
</library>
```

//@lang

XPath

```
<library>
 <title>My book collection</title>
 <book>
    <title lang="en">Harry Potter</title>
    <price>1234</price>
 </book>
 <book>
    <title lang="hu">Data-driven systems</title>
    <price>5678</price>
 </book>
</library>
```

/library/book[1]

XPath

```
<library>
 <title>My book collection</title>
 <book>
    <title lang="en">Harry Potter</title>
    <price>1234</price>
 </book>
 <book>
    <title lang="hu">Data-driven systems</title>
    <price>5678</price>
 </book>
</library>
```

/library/book[price>5000]

XPath

```
<library>
 <title>My book collection</title>
 <book>
    <title lang="en">Harry Potter</title>
    <price>1234</price>
 </book>
 <book>
    <title lang="hu">Data-driven systems</title>
    <price>5678</price>
 </book>
</library>
```

/library/book/price[text()]

Handling semi-structured data

JSON

JSON

- JavaScript Object Notation
  - Not much to do with JavaScript!
- Compact, readable, textual representation.
- A single in-memory object is a single JSON document.
- Building blocks:
  - *Object*
    - List of key-value pairs
  - Array
    - List of values
  - Value
    - Text, number, true/false, null, *object*, array

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212,555-1234"
    },
    {
      "type": "mobile",
      "number": "123,456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

JSON - problems

- No comment
- Cannot contain a byte order mark
  - No need, the first character code is always smaller than 128
- Does not specify representation of frequent data types
  - E.g. date
  - The standard does not specify.
  - Requires additional information for parsing.
- Security risk.
  - Typical, but must be avoided: JSON result is parsed using JavaScript `eval()` method

JSON Schema

- Schema description
  - Just like XSD is for XML
- JSON document itself

JSON Schema

```
{ "$schema": "http://json-schema.org/schema#",
  "title": "Product",
  "type": "object",
  "required": ["id", "name"],
  "properties": {
    "id": {
      "type": "number",
      "description": "Product identifier"
    },
    "name": {
      "type": "string",
    },
    "stock": {
      "type": "object",
      "properties": {
        "warehouse": { "type": "number"  },
        "retail": { "type": "number" }
      }
    } } }
```

JSON – when to use

- Backend - thin-client frontend communication
  - Compact, small.
    - Little network traffic, good for mobile devices.
  - Can be parsed by JavaScript.
    - In web applications.
- REST
  - See later.
- JSON in databases
  - MS SQL Server 2016, Oracle Server 12c
  - **MongoDB → we will see soon**

.NET

- System.Text.Json

```
var weatherForecast = new WeatherForecast {
     Date = DateTime.Parse("2019-08-01"),
     TemperatureCelsius = 25,
     Summary = "Hot" };

string jsonString =
     JsonSerializer.Serialize(weatherForecast);


WeatherForecast? weatherForecast =
     JsonSerializer.Deserialize<WeatherForecast>(jsonString);
```

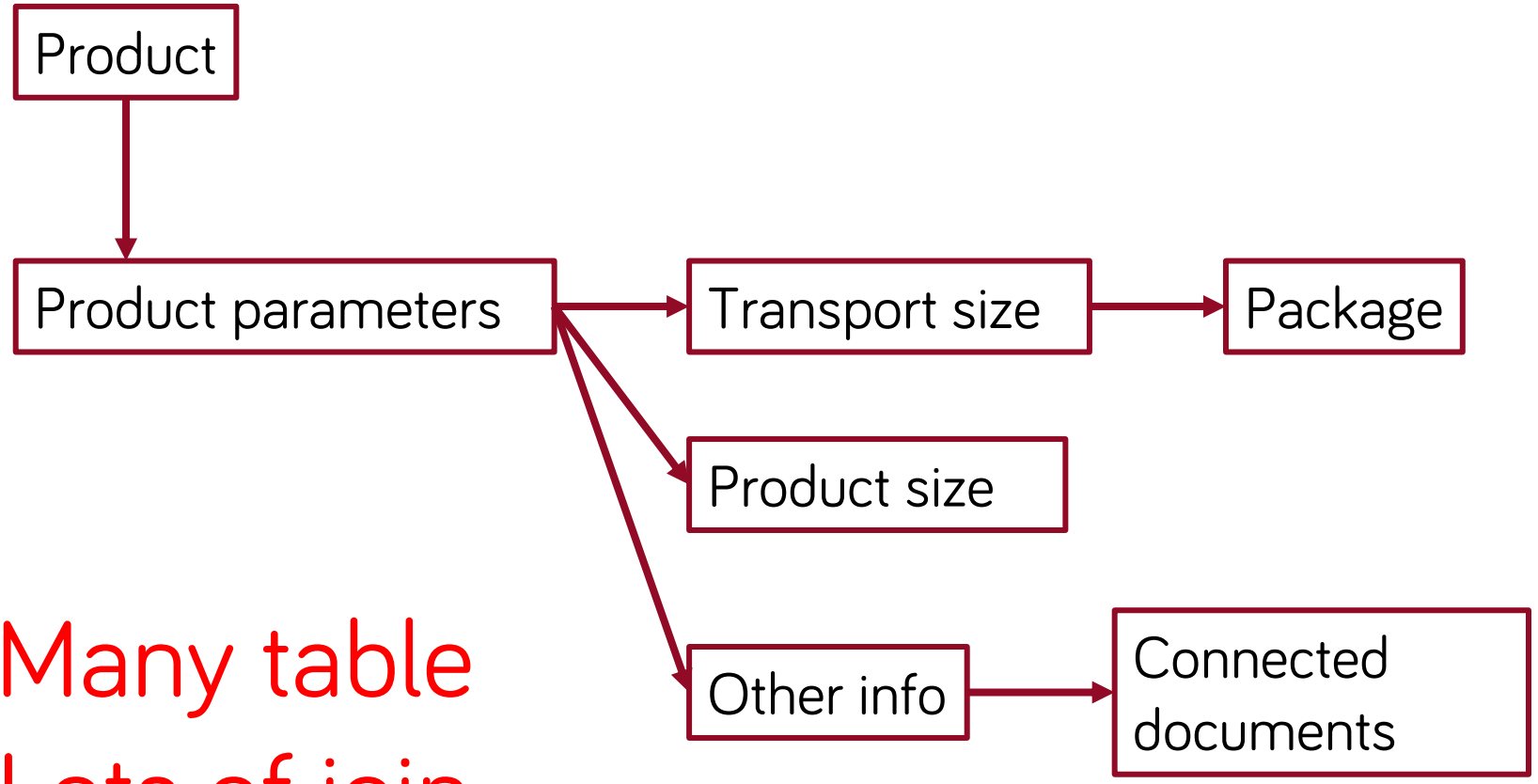
XML <-> JSON

	XML	JSON
Data types	Not defined.	Few scalar, object, array.
Arrays	Not specified, but can be represented.	Defined by the format.
Objects	Not specified, multiple ways to represent.	Defined by the format.
Null value	xsi:nil (+namespace import)	Supports
Comment	Supports	No
Namespace	Supports	No
Representation, formatting	Ambiguous	Unambiguous, except for date
Size	Longer	Compact
Parse in JavaScript	Complex	Simple
Required knowledge	Multiple technologies	JavaScript

XML handling in relational data bases

Handling semi-structured data - sample use-case

Handling semi-structured data - relational example



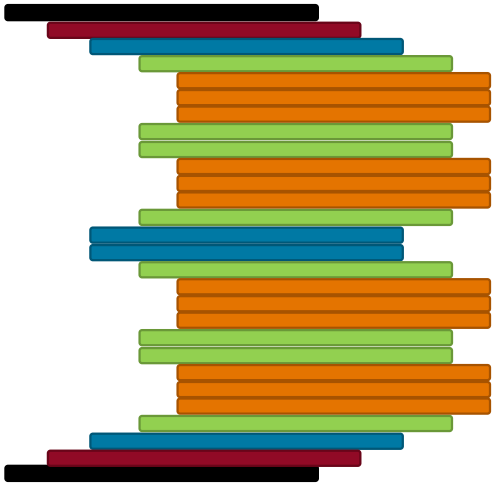
Many table
Lots of join

Semi-structured data - XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<product>
   <transport_size>
      <number_of_packages>2</<number_of_packages>
         <package id="1">
            <size>
               <width>80</width>
               <height>20</height>
               <depth>40</depth>
            </size>
         </package>
      <package id="2">...</package>
   </transport_size>
   <product_size>...</product_size>
   <additional_information>...</additional_information>
</product>
```

Semi-structured data in databases

- Existing xml data.
- Unknown data without format specification.
- Data coming from an external system in such form.
- Data that is only stored, not altered.
- Deeply nested data structures.



Semi-structured data in databases

ID	Name	…	
123	Wardrobe	…	<?xml>…
456	Sofa	…	<?xml>…

Storing XML data in relational data bases

- Relational databases supporting XML data.
  - Microsoft SQL, Oracle, PostgreSql, …
- Xml data ***besides*** relational data.
- Relational is the main content.
- Xml data is mapped to relational entities.
  - E.g. product details as xml

Data types in MSSQL

- nvarchar(max)
  - Textual data, contents not checked.
  - Keeps the content as-is.
  - Can be converted runtime (expensive).
- xml
  - Must be valid xml.
  - A schema can be defined; content is automatically validated.
  - Searchable, e.g. the value of a node.
  - Can be altered within the database.
  - Index can be defined.

```
create table Product(
    Name nvarchar(100),
    Description XML )
```

Index for xml columns

- To be used only when searching for content within xml.
  - Index is not used when the entire xml content is retrieved.
- Two types of indices:
  - Primary: indexes the entire content
    - Can only be one of such.
    - If the xml column is indexes, one such index must exist.

```
CREATE PRIMARY XML INDEX idxname on Table(Col)
```

  - Secondary: defined on a specific part of the xml document
    - Multiple ones can exist.

```
CREATE XML INDEX idxname2 ON Table(Col)
USING XML INDEX idxname FOR VALUE;
```

Defining a schema for an xml column

- The system validates the content automatically.
  - Acts like an integrity criteria.
- Also used by query optimization.
- Optional.

Query

- query(XQuery)

```
> select
  Description.query('/product/num_of_packages')
  from Product
    <num_of_packages>1</num_of_packages>
```

- value(XQuery, SQLType)

```
> select
  Description.value('(/product/num_of_packages)[1]','
  int') from Product
    1
```

- exist (XQuery)

```
> select Name from Producr
  where Description.exist('/product/num_of_packages
  eq 2')=1
```

Manipulation

- modify()

```
update Product
set Description.modify(
'replace value of
(/product/num_of_packages/text())[1] with "2"')
where ID=8

update Product
set Description.modify(
'insert <a>1</a> after (/product)[1]')
where ID=8

update Product
set Description.modify('delete /product/a')
where ID=8
```

FOR XML

- Convert the result of a query to XML

```
select ID, Name from Customer
for xml auto
```

```
<Customer ID="1" Name=”John Doe" />
<Customer ID="2" Name=”Jane Doe" />
<Customer ID="3" Name=”..." />
```

JSON handling in relational data bases

MS SQL JSON support

- There is no special data type like XML, it stores JSON in NVARCHAR
  - There is no special index either
- Functions
  - **ISJSON**: is the json of the given string
  - **JSON_VALUE**: extracts a scalar value from json
  - **JSON_QUERY**: returns a json chunk (object or array in json)
  - **JSON_MODIFY**: similar to xml.modify
    - update / delete / insert
  - **OPENJSON**: Returns SQL rows from json
  - **FOR JSON**: returns the result of the query in json format

Query sample

- Relational and json data can be used together in queries

```
SELECT Name,
    Surname,
    JSON_VALUE(jsonCol, '$.info.address.PostCode') AS PostCode,
    JSON_VALUE(jsonCol, '$.info.address."Address Line 1"')
        + ' ' + JSON_VALUE(jsonCol, '$.info.address."Address Line 2"') AS Address,
    JSON_QUERY(jsonCol, '$.info.skills') AS Skills
FROM People
WHERE ISJSON(jsonCol) > 0
    AND JSON_VALUE(jsonCol, '$.info.address.Town') = 'Belgrade'
    AND STATUS = 'Active'
ORDER BY JSON_VALUE(jsonCol, '$.info.address.PostCode');
```

Query from json

- Using a variable (or stored procedure parameter)

```
DECLARE @json NVARCHAR(MAX);

SET @json = N'[
  {"id": 2, "info": {"name": "John", "surname": "Smith"}, "age": 25},
  {"id": 5, "info": {"name": "Jane", "surname": "Smith"}, "dob": "2005-11-04T12:00:00"}
]';

SELECT *
FROM OPENJSON(@json) WITH (
    id INT 'strict $.id',
    firstName NVARCHAR(50) '$.info.name',
    lastName NVARCHAR(50) '$.info.surname',
    age INT,
    dateOfBirth DATETIME2 '$.dob'
    );
```

ID	firstName	lastName	age	dateOfBirth
2	John	Smith	25	
5	Jane	Smith		2005-11-04T12:00:00

Export to json

- Format a query into json string

```
SELECT id,
    firstName AS "info.name",
    lastName AS "info.surname",
    age,
    dateOfBirth AS dob
FROM People
FOR JSON PATH;
```

```
[
  {
    "id": 2,
    "info": {
      "name": "John",
      "surname": "Smith"
    },
    "age": 25
  },
  {
    "id": 5,
    "info": {
      "name": "Jane",
      "surname": "Smith"
    },
    "dob": "2005-11-04T12:00:00"
  }
]
```