Course A7B36DBS: **Database Systems**

Lecture 05:

Embedded SQL, SQL/XML

Martin Svoboda

Faculty of Electrical Engineering, Czech Technical University in Prague

Outline

- **SQL**
  - Embedded SQL
    - Internal database applications
    - **Stored procedures, cursors, triggers**
  - External applications
    - Standardized interfaces
  - **SQL/XML**
    - Manipulation with XML data

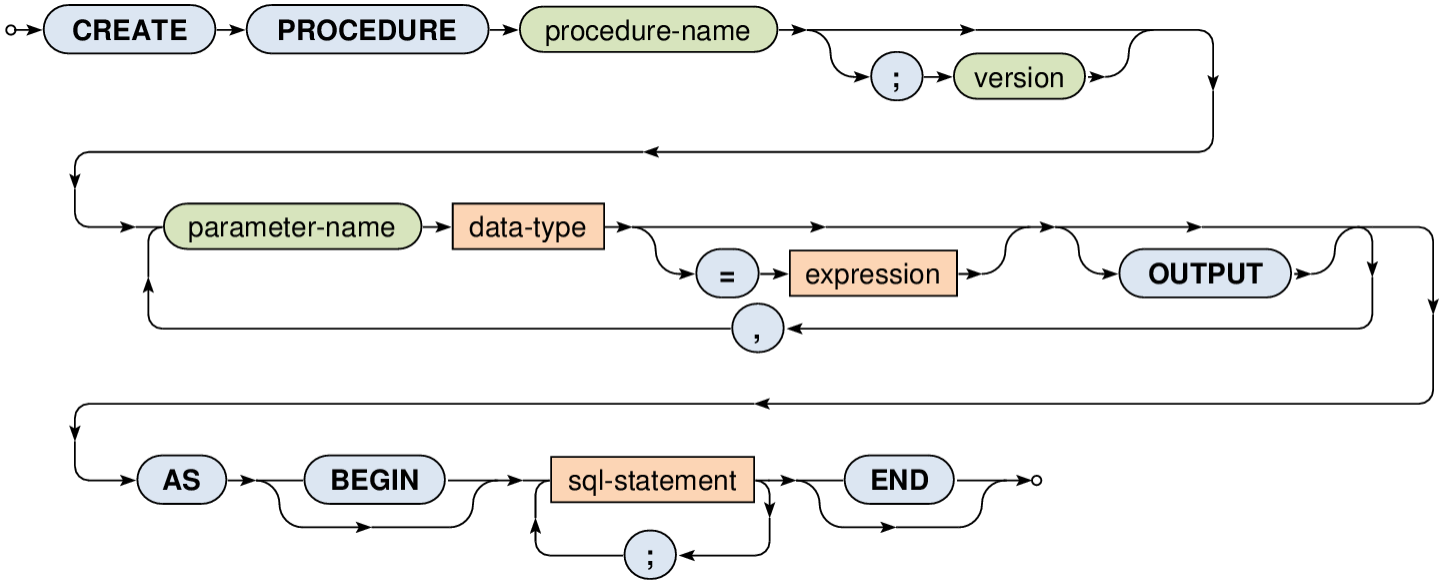
Embedded SQL

Embedded SQL

- **Internal database applications**
  - Procedural extensions of SQL
  - Proprietary solutions
    - Transact SQL (T-SQL) – Microsoft SQL Server
    - PL/SQL – Oracle Database
  - Available constructs
    - Control statements: if then else, for, while, switch
    - **Stored procedures**
    - **Cursors** – iterative scanning of tables
    - **Triggers** – general integrity constraints
    - …

Stored Procedures

- **CREATE PROCEDURE**
  - Definition of a stored procedure
    - Allows us to reuse procedural SQL code

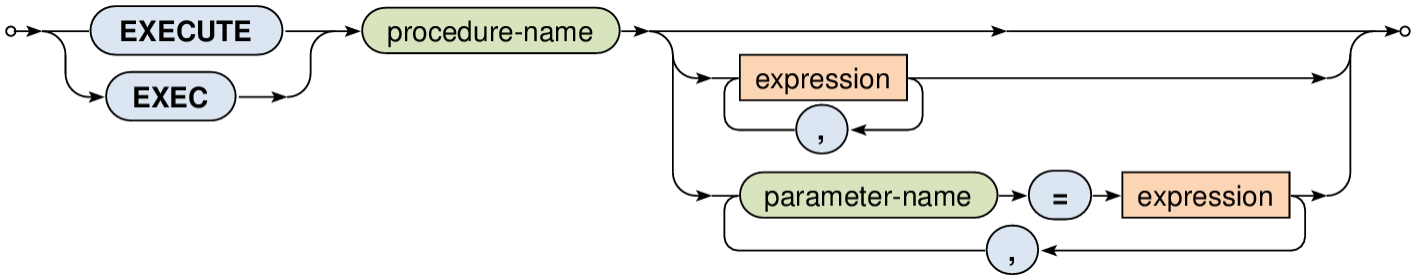


Stored Procedures

- Notes…
  - **Version** – number describing the procedure version
    - We can have multiple versions of procedures having exactly the same name
  - **Parameter name** – must begin with @
  - **Default values** for parameters
  - **OUTPUT** – declaration of an output parameter
    - Otherwise a given parameter is an input one

Stored Procedures

- **Procedure call**
  - Two styles of passing parameters:
    - Without names – we must respect the original order
    - With names



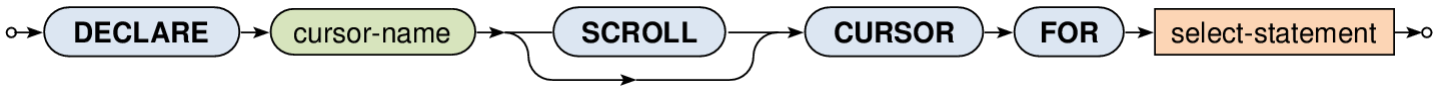
Stored Procedures: Example

```
CREATE PROCEDURE Payment
  @accountSource VARCHAR,
  @accountTarget VARCHAR,
  @amount INTEGER = 0
AS
BEGIN
   UPDATE Accounts
      SET balance = balance - @amount
      WHERE (account = @accountSource);
   UPDATE Accounts
      SET balance = balance + @amount
      WHERE (account = @accountTarget);
END

EXEC Payment "21-87526287/0300", "78-9876287/0800", 25000;
```

Cursors

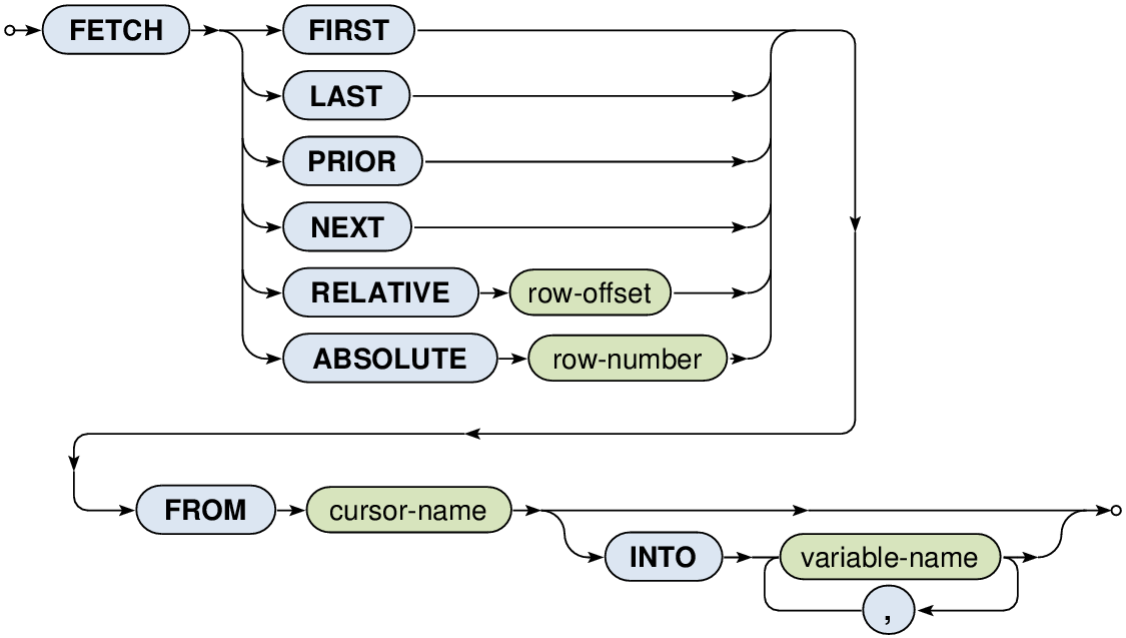
- **Cursor declaration**
  - Database cursor is a control structure that allows us to traverse over the rows of a selected table



  - **SCROLL** cursor alternative
    - All fetch options (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE) are available
    - Otherwise only the NEXT fetch option is permitted

Cursors

- **Data retrieval**
  - Fetch options – which row should be provided?



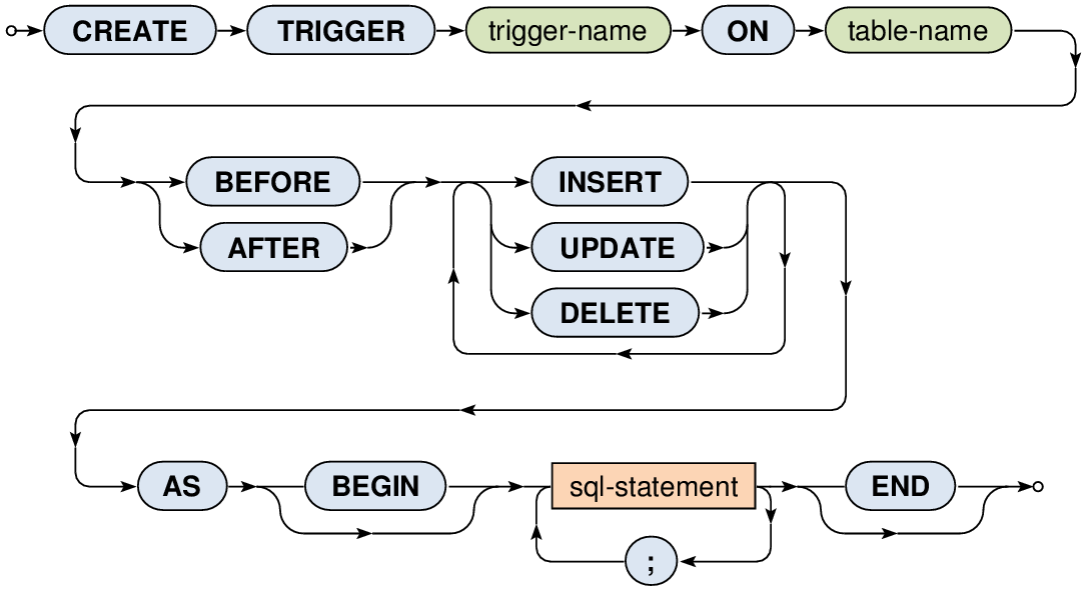
  - **INTO**: local variables into which the row should be stored

Cursors: Example

```
DECLARE C CURSOR FOR
   SELECT * FROM Accounts;
BEGIN
   OPEN C;
   DECLARE @account VARCHAR, @balance INT;
   FETCH NEXT FROM C INTO @account, @balance;
   WHILE @@FETCH_STATUS = 0
   BEGIN
      EXEC Payment IRS_Account, @account, @balance * 0.01
      FETCH NEXT FROM C;
   END;
   CLOSE C;
   DEALLOCATE C;
END
```

Triggers

- **CREATE TRIGGER**
  - Trigger is a procedure that is automatically executed as a response to certain events (INSERT, UPDATE, DELETE)
    - Used for maintaining complex integrity constraints



External Database Applications

External Applications

- **Standardized interfaces** for working with RDBMS from standalone applications:
  - **ODBC** (Open DataBase Connectivity)
    - 1992, Microsoft
  - **JDBC** (Java DataBase Connectivity)
    - Using ODBC, or native driver/protocol, network driver
  - **ADO.NET library** (Active Data Objects .NET)
    - OLE DB, ODBC, or direct drivers to MS SQL Server, Oracle
- Native drivers

Java and JDBC

```
import java.sql.*;

Class.forName("com.jdbcvendor.JdbcDriver");
Connection c = DriverManager.getConnection(
   "jdbc:jdbcvendor:database",
   "myLogin",
   "myPassword"
);

Statement s = c.createStatement();
try {
   s.executeUpdate(
      "INSERT INTO MyTable VALUES ('my name')"
   );
} finally { s.close(); }
```

SQL/XML

XML Documents: Example

```
<?xml version="1.0"?>
<library>
   <book id="1" catalogue="c1" language="en">
      <title>XPath</title>
      <author>John</author>
      <author>Peter</author>
   </book>
   <book id="2" catalogue="c1">
      <title>XQuery</title>
      <price>25</price>
   </book>
   <book id="3" catalogue="c2" language="en">
      <title>XSLT</title>
      <author>John</author>
   </book>
</library>
```

Introduction

- **SQL/XML**
  - **Extension to SQL for XML data**
    - XML Datatype
    - Constructs
      - Functions, constructors, mappings, XQuery embedding, …
- Standards
  - **SQL:2011-14** (ISO/IEC 9075-14:2011)
    - Older versions 2003, 2006, 2008

Example

- **Table**: books

id	catalogue	title	details	language
1	c1	XPath	<author>John</author> <author>Peter</author>	en
2	c1	XQuery	<price>25</price>	NULL
3	c2	XSLT	<author>John</author>	en

- **Table**: languages

code	name
en	English
cs	Czech

Example

- **Query**

```
SELECT
   id,
   XMLELEMENT(
      NAME "book",
      XMLELEMENT(NAME "title", title),
      details
   ) AS book
FROM books
WHERE (language = "en")
ORDER BY title DESC
```

Example

- **Result**

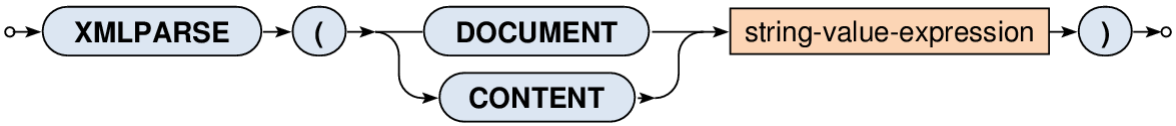
id	book
3	<book> <title>XSLT</title> <author>John</author> </book>
1	<book> <title>XPath</title> <author>John</author> <author>Peter</author> </book>

XML Datatype

- Traditional types
  - BLOB, CLOB, VARCHAR, …
- **Native XML type**
  - Collection of information items
    - Based on XML Information Set (**XML Infoset**)
      - Elements, attributes, processing instructions, …
      - But we also allow fragments without right one root element
        - » This means that XML values may not be XML documents
  - NULL

Parsing XML Values

- XMLPARSE
  - **Creates an XML value from a string**
    - DOCUMENT – well-formed document with right one root
    - CONTENT – well-formed fragment

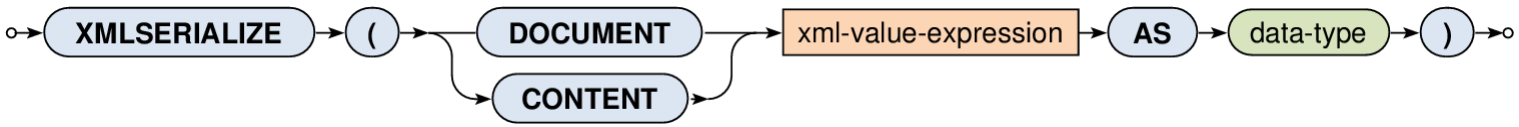



```
SELECT XMLPARSE(
   DOCUMENT "<book><title>XPath</title></book>"
) AS result
```

result
<book> <title>XPath</title> </book>

Serializing XML Values

- XMLSERIALIZE
  - **Exports an XML value to a string**




```
SELECT
   id, title,
   XMLSERIALIZE(CONTENT details AS VARCHAR(100)) AS export
FROM books
```

id	title	export
1	XPath	<author>John</author><author>Peter</author>
...	...	...

Well-Formedness Predicate

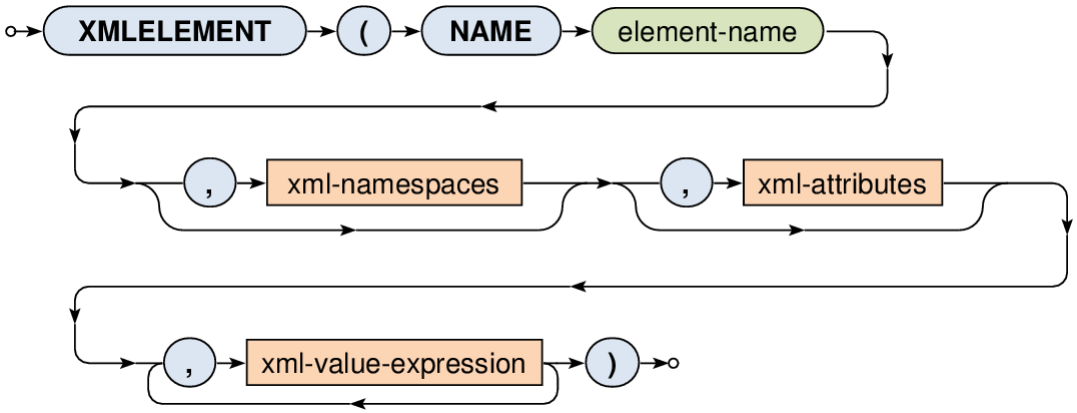
- IS DOCUMENT
  - **Tests whether an XML value is an XML document**
    - Returns `TRUE` if there is right one root element
    - Otherwise `FALSE`

Constructors

- Functions for construction of XML values…
  - **XMLELEMENT** – elements
  - **XMLNAMESPACES** – namespace declarations
  - **XMLATTRIBUTES** – attributes
  - **XMLCOMMENT** – comments
  - **XMLPI** – processing instructions
  - **XMLFOREST** – sequences of elements
  - **XMLCONCAT** – concatenations of values
  - **XMLAGG** – aggregates

Elements

- XMLELEMENT
  - **Creates an XML element** with a given name and…
    - optional **namespace declarations**
    - optional **attributes**
    - optional **element content**



Elements: Example 1

```
SELECT
   id,
   XMLELEMENT(NAME "book", title) AS result
FROM books
ORDER BY id
```

id	result
1	<book>XPath</book>
2	<book>XQuery</book>
3	<book>XSLT</book>

Elements: Example 2: Subelements

```
SELECT
   id,
   XMLELEMENT(
      NAME "book",
      XMLELEMENT(NAME "title", title),
      XMLELEMENT(NAME "language", language)
   ) AS records
FROM books
```

id	records
1	<book> **<title>XPath</title>** **<language>en<language>** </book>
…	…

Elements: Example 3: Mixed Content

```
SELECT
   id,
   XMLELEMENT(
      NAME "info",
      "Book ", XMLELEMENT(NAME "title", title),
      " with identifier equal to", id, "."
   ) AS description
FROM books
```

id	description
1	<info> **Book <title>XPath</title> with identifier equal to 1.** </info>
…	…

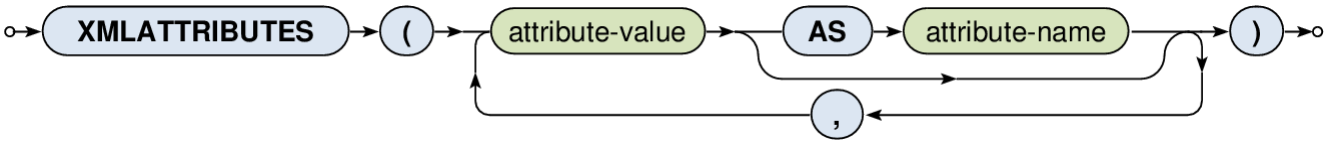
Elements: Example 4: Subqueries

```
SELECT
   id,
   XMLELEMENT(NAME "title", title) AS book,
   XMLELEMENT(
      NAME "language",
      (SELECT name FROM languages WHERE (code = language))
   ) AS description
FROM books
```

id	book	description
1	<title>XPath</title>	<language>**English**</language>
…	…	…

Attributes

- XMLATTRIBUTES
  - **Creates a set of attributes**
  - Input: list of values
    - Each value must have an **explicit / implicit name**
      - It is used as a name for the given attribute
      - Implicit names can be derived, e.g., from column names
  - Output: XML value representing created attributes



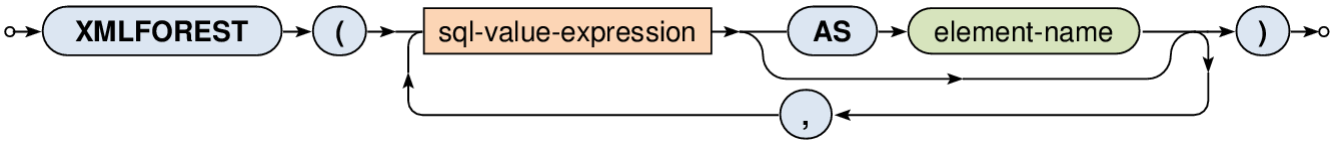
Attributes: Example

```
SELECT
   id,
   XMLELEMENT(NAME "book",
      XMLATTRIBUTES(
         language, catalogue AS "location"
      ),
      XMLELEMENT(NAME "title", title)
   ) AS book
FROM books
```

id	book
1	<book **language="en" location="c1"**> <title>XPath</title> </book>
…	…

Element Sequences

- XMLFOREST
  - **Creates a sequence of XML elements**
  - Input: list of SQL values
    - Individual content expressions evaluated to `NULL` are ignored
    - If all the expressions are evaluated to `NULL`, then `NULL` is returned
  - Each content value must have an **explicit / implicit name**
    - It is used as a name for the given element
  - Output: XML value (forest of elements)



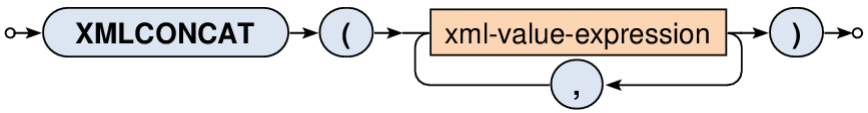
Element Sequences: Example

```
SELECT
   id,
   XMLFOREST(
      title, language, catalogue AS location
   ) AS book
FROM books
```

id	book
1	<title>XPath</title> <language>en</language> <location>c1</location>
2	<title>XQuery</title> <location>c1</location>
…	…

Concatenation

- XMLCONCAT
  - **Creates a sequence from a list of values**
  - Input: list of XML values
    - Individual content expressions evaluated to `NULL` are ignored
    - If all the expressions are evaluated to `NULL`, then `NULL` is returned
  - Output: XML value (forest)



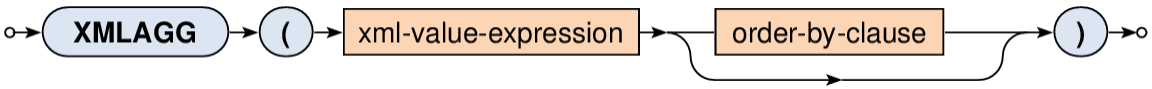
Concatenation: Example

```
SELECT
   id,
   XMLCONCAT(
      XMLELEMENT(NAME "book", title),
      details
   ) AS description
FROM books
```

id	description
1	<book>XPath</book> <author>John</author> <author>Peter</author>
…	…

XML Aggregation

- XMLAGG
  - **Aggregates rows within a given super row**
    - I.e. acts as a standard aggregate function (like SUM, AVG, …)
  - Input: rows within a given super row
    - These rows can first be optionally sorted (**ORDER BY**)
    - For each row an XML value is generated as described
      - Individual rows evaluated to `NULL` values are ignored
    - All the generated XML values are then concatenated
      - If all the rows are evaluated to `NULL`, then `NULL` is returned
  - Output: XML value (forest)



XML Aggregation: Example

```
SELECT
   catalogue,
   XMLAGG(
      XMLELEMENT(NAME "book", XMLATTRIBUTES(id),
         title)
      ORDER BY id
   ) AS list
FROM books
GROUP BY catalogue
```

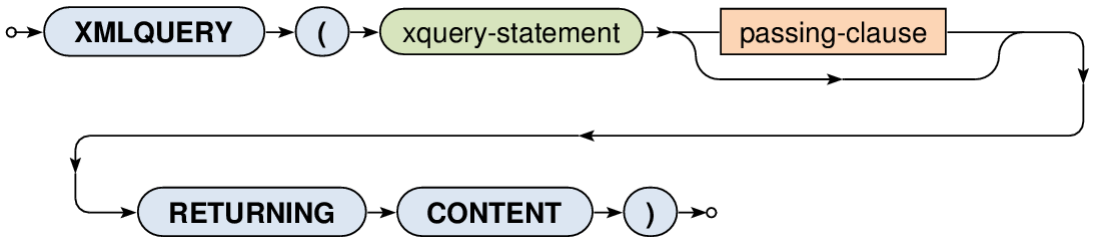
catalogue	list
c1	<book id="1">XPath</book> <book id="2">XQuery</book>
c2	<book id="3">XSLT</book>

Querying

- Query constructs
  - Based on XQuery language
  - **XMLQUERY** – returns query result
    - Usually in SELECT clauses
  - **XMLTABLE** – decomposes query result into a table
    - Usually in FROM clauses
  - **XMLEXISTS** – tests query result nonemptiness
    - Usually in WHERE clauses

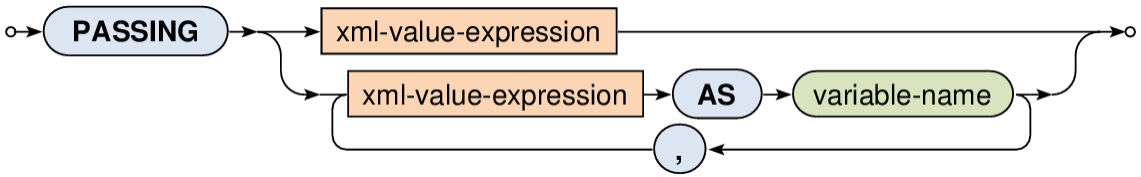
XQuery Statements

- XMLQUERY
  - **Evaluates an XQuery statement and returns its result**
  - Input:
    - XML values declared in an optional **PASSING** clause
  - Output: XML value



XQuery Statements

- XMLQUERY
  - Input data
    - When **only one input value** is specified…
      - its content is accessible via `/` within the XQuery statement
    - When **one or more named variables** are specified…
      - their content is accessible via `$variable-name/`



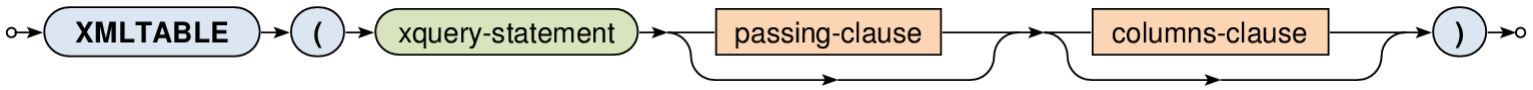
XQuery Statements: Example

```
SELECT
   id, title,
   XMLQUERY(
      "<authors>{ count($data/author) }</authors>"
      PASSING details AS data
      RETURNING CONTENT
   ) AS description
FROM books
```

id	title	description
1	XPath	<authors>2</authors>
…	…	…

XML Tables

- XMLTABLE
  - **Decomposes an XQuery result into a virtual table**
  - Output:
    - When **COLUMNS** clause is specified…
      - Table containing the XQuery result being shredded into individual rows and columns according to the description
    - Otherwise…
      - Table with one row and one column with the XQuery result represented as an XML value



XML Tables: Example 1

```
SELECT
   id, title, result.*
FROM
   books,
   XMLTABLE(
      "<authors>{ count($data/author) }</authors>"
      PASSING books.details AS data
   ) AS result
```

id	title	result
1	XPath	<authors>2</authors>
…	…	…

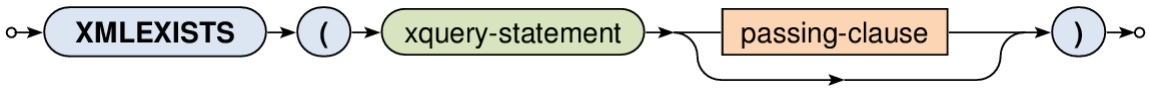
XML Tables: Example 2

```
SELECT
   id, title, result.count
FROM
   books,
   XMLTABLE(
      "<authors>{ count($data/author) }</authors>"
      PASSING books.details AS data
      COLUMNS
         count INTEGER PATH "authors/text()"
   ) AS result
```

id	title	count
1	XPath	**2**
…	…	…

Exists Predicate

- XMLEXISTS
  - **Tests an XQuery statement result for nonemptiness**
  - Output: Boolean value
    - Returns `TRUE` for result sequences that are not empty
    - Otherwise `FALSE`



Exists Predicate: Example

```
SELECT books.*
FROM books
WHERE
   XMLEXISTS(
      "/author"
      PASSING details
   )
```

id	catalogue	title	details	language
1	c1	XPath	<author>John</author> <author>Peter</author>	en
3	c2	XSLT	<author>John</author>	en