course:
Database Systems (NDBI025)
SS2018/19

lecture 5:

SQL – embedded SQL, SQL/XML

doc. RNDr. Tomáš Skopal, Ph.D.
doc. RNDr. Irena Mlýnková, Ph.D.
RNDr. Michal Kopecký, Ph.D.

Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague

Today's lecture outline

- embedded SQL – „internal" database applications
  - stored procedures
  - cursors
  - triggers
- SQL/XML
  - XML data type + functions integrated into SQL SELECT

Programming in embedded SQL

- procedural extensions of SQL
  - std. SQL is a subset (i.e., that's why embedded)
  - MS SQL Server – Transact SQL (T-SQL)
  - Oracle – PL/SQL
- benefits
  - controlling statements (if-then, for, while)
    - cannot be just scripted
  - cursors (iterative scan of tables)
  - smaller networking overhead (code on server), pre-compiled
  - triggers – general integrity constraints
  - better security (access rights control for server code)
- cons
  - proprietary extensions, cannot be simply transferred
  - SQL 1999 standard, but not respected by industry much

Structure (MS SQL Server)

DECLARE @var_name data_type
sql_statement
BEGIN … **END**
…

E.g.:

DECLARE @avg_age FLOAT
BEGIN
SELECT @avg_age = AVG(age) FROM Employee
END

Stored procedures (MS SQL Server)

CREATE PROCEDURE *procname* [; *number*]
[*declaration_parameter* [, ...]]
[WITH RECOMPILE]
AS commands [;]

- parameter declaration
  - *@name type* [= *expression*] [OUT[PUT]]
    - OUT[PUT] parameter is output
- *number* allows multiple versions of the same procedure
- procedure call
  - EXEC[UTE] procname [expression [, ...]]
    - parameters are passed w.r.t. order
  - EXEC[UTE] procname [@name=expression [, ...]]
    - parameters are passed w.r.t. names

Stored procedures, example

```
CREATE PROCEDURE Payment
      @accSource VARCHAR(25),
      @accTarget VARCHAR(25),
      @amount INTEGER = 0
    AS
    BEGIN
          UPDATE Accounts SET balance = balance - @amount
           WHERE account=@accSource;

          UPDATE Accounts SET balance = balance + @amount
           WHERE account=@accTarget;
    END

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

Stored procedures (MS SQL Server)

CREATE FUNCTION *funcname* [; *number*]
([*declaration_parameter* [, ...]]) **RETURNS** type
[WITH RECOMPILE]
AS commands [;]

- parameter declaration
  - *@name type* [= *expression*] [OUT[PUT]]
    - OUT[PUT] parameter is output
- *number* allows multiple versions of the same procedure
- function use
  - schemaname.funcname([expression [, ...]])
    - parameters are passed w.r.t. order
  - schemaname.funcname([@name=expression [, ...]])
    - parameters are passed w.r.t. names

Stored procedures, example

```
CREATE FUNCTION AccBalance(
      @acc VARCHAR(25)
      ) RETURNS INTEGER
     AS
     DECLARE @ret INTEGER;
     BEGIN
           SELECT @ret =balance
            FROM Accounts
            WHERE account=@acc;

           RETURN @ret;
     END

SELECT MySchema.AccBalance(account) AS bal FROM …
```

Cursors (MS SQL Server)

- declaration
  - C [SCROLL] **CURSOR FOR**
    SELECT …;
- data retrieval
  - **FETCH**
    {NEXT | PRIOR | ABSOLUTE n | RELATIVE n | LAST | FIRST}
    FROM C
    [**INTO** @variable [, …]]
  - If cursor not declared using SCROLL, only NEXT is allowed

Cursors, example (tax payment)

```
DECLARE
    Cur CURSOR FOR
        SELECT *
        FROM Accounts;
    BEGIN
        OPEN Cur
        DECLARE @acc varchar(25), @bal int;
        FETCH NEXT FROM Cur INTO @acc, @bal;
        WHILE @@FETCH_STATUS=0
        BEGIN
         Payment(@acc, '21-87526287/0300', @bal*0.01)
         FETCH NEXT FROM Cur INTO @acc, @bal;
        END;
        CLOSE Cur;
        DEALLOCATE Cur;
    END
```

Triggers – DML triggers

- event-executed stored procedure (table/view event)
- allows to extend integrity constraint logics
  - ***inserted, deleted*** – logical tables with the same structure as the table the trigger is bound on

```
CREATE TRIGGER trigger_name ON { table | view }
[ WITH ENCRYPTION ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
AS
[{IF UPDATE ( column ) [{ AND | OR } UPDATE ( column ) ] ...
  | IF ( COLUMNS_UPDATED (  bitwise_operator updated_bitmask
)}]
sql_statement [...]
```

DML triggers (example)

```
CREATE TRIGGER LowCredit ON PurchaseOrderHeader
AFTER INSERT
AS
DECLARE @error_count int

SELECT @error_count = count (*)
    FROM inserted i JOIN Purchasing.Vendor v on v.VendorID = i.VendorID
    WHERE v.CreditRating=5

IF @error_count > 0
BEGIN
    RAISERROR ('Vendor''s credit rating is too low to accept new purchase orders.', 16, 1)
    ROLLBACK TRANSACTION
END
```

DML triggers (example)

```
CREATE TRIGGER LowCredit ON PurchaseOrderHeader
AFTER INSERT
AS
IF EXISTS(
    SELECT *
    FROM inserted i JOIN Vendor v on (v.VendorID = i.VendorID)
    WHERE v.CreditRating=5
    )
BEGIN
    RAISERROR ('Vendor''s credit rating is too low to accept new purchase orders.', 16, 1)
    ROLLBACK TRANSACTION
END
```

DDL triggers

CREATE TRIGGER trigger_name **ON** { ALL SERVER | DATABASE }
[WITH <ddl_trigger_option> [,...n]]
{ FOR | AFTER } { event_type | event_group } [,...n] **AS**
{sql_statement [;] [,...n] | EXTERNAL NAME < method specifier > [;] }
<ddl_trigger_option> ::= [ENCRYPTION] [EXECUTE AS Clause]
    <method_specifier> ::= assembly_name.class_name.method_name

DDL triggers (example)

```
CREATE TRIGGER safety ON DATABASE FOR
    DROP_SYNONYM
AS
    RAISERROR (
       'You must disable Trigger "safety" to drop synonyms!',
       10, 1
    )
    ROLLBACK
```

What is SQL/XML

- An extension of SQL for XML data (SQL 2003)
  - New built-in data type called XML
  - Querying over XML data
- Note: SQL/XML ≠ SQLXML
  - SQLXML is Microsoft technology in MS SQL Server (not standard)
  - Similar strategy, but different approach
  - Not a standard
- The key aspect is an XML value
  - Intuitively: an XML element or a set of XML elements
  - Its semantics results from XML Infoset data model
    - Standard of XML data model = tree consisting of nodes representing elements, attributes, text values, ...
    - graph formalism for XML

SQL/XML – XML Data Publishing

- Generating of XML data (of type XML) from relational data
  - XMLELEMENT – creating an XML element
  - XMLATTRIBUTES – creating XML attributes
  - XMLFOREST – creating a sequence of XML elements
  - XMLCONCAT – concatenation of XML values into a single value
  - XMLAGG – creating a sequence of elements from a group of rows

SQL/XML – XMLELEMENT

```
Employees (id, first, surname, dept, start)
```

- Creating an XML value for:
  - Name of an element
  - (Optional) list of attribute declarations
  - (Optional) list of expressions declaring element content

```
SELECT E.id,
  XMLELEMENT ( NAME "emp",
    E.first || ' ' || E.surname ) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<emp>George Clooney</emp>
...	...

XMLELEMENT – subelements

```
SELECT E.id,
  XMLELEMENT ( NAME "emp",
    XMLELEMENT ( NAME "name",
      E.first || ' ' || E.surname ),
    XMLELEMENT ( NAME "date", E.start )
  ) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<emp> <name>George Clooney</name> <date>2000-05-24</date> </emp>
...	...

SQL/XML – XMLATTRIBUTES

```
SELECT E.id,
  XMLELEMENT ( NAME "emp",
    XMLATTRIBUTES ( E.id AS "empid" ),
    E.first || ' ' || E.surname ) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<emp empid="1001">George Clooney</emp>
...	...

SQL/XML – XMLFOREST

```
SELECT E.id,
  XMLELEMENT ( NAME "emp",
    XMLFOREST (
      E.first || ' ' || E.last AS "name",
      E.start AS "date" ) ) AS xvalue
FROM Employees E
WHERE ...
```

id	xvalue
1001	<emp> <name>George Clooney</name> <date>2000-05-24</date> </emp>
...	...

SQL/XML – XMLAGG

- XMLAGG is an aggregation function combined with GROUP BY
  - Similarly to SUM, AVG etc.
- XMLAGG accepts only XML expressions (values)
- The expression is evaluated for each row in group G created by the GROUP BY expression
- The results are concatenated into a single resulting value
- The result can be sorted using ORDER BY

SQL/XML – XMLAGG

```
SELECT XMLELEMENT (
  NAME "dept",
  XMLATTRIBUTES (E.dept AS "name"),
  XMLAGG (
    XMLELEMENT (NAME "emp", E.surname)
    ORDER BY E.surname )
  ) AS xvalue
FROM Employees E
GROUP BY E.dept
```

xvalue
<dept name="hr"> <emp>Clooney</emp> <emp>Pitt</emp> </dept>
<dept name="accountant"> … </dept>

XML Type

- Type XML can be used at same places as SQL data types (e.g., NUMBER, VARCHAR, …)
  - Column type, SQL variable, …
- The XML type can be
  - Queried (XMLQUERY)
  - Transformed into relational data (XMLTABLE)
  - Tested (XMLEXISTS)

Sample Data – Table EmpXML

id	ColEmpXML
1001	<emp> <first>George</first> <surname>Clooney</surname> <date>2000-05-24</date> <dept>hr</dept> </emp>
1006	<emp> <first>Brad</first> <surname>Pitt</surname> <date>2001-04-23</date> <dept>hr</dept> </emp>
…	…

XMLQUERY

```
SELECT
  XMLQUERY (                                  XQuery query
    'for $p in $col/emp return $p/surname',
    PASSING EmpXML.ColEmpXML AS "col"
    RETURNING CONTENT NULL ON EMPTY ) AS result
FROM EmpXML WHERE …
```

result
<surname>Clooney</surname>
<surname>Pitt</surname>
…

XMLTABLE

```
SELECT result.*
  FROM EmpXML, XMLTABLE (
    'for $p in $col/emp return $p',
    PASSING EmpXML.ColEmpXML AS "col"
    COLUMNS firstname VARCHAR(40) PATH 'first'
                      DEFAULT 'unknown',
            lastname  VARCHAR(40) PATH 'surname'
  ) AS TableResult
```

XQuery query

XPath expressions over XML data returned by the query

Assumption: We do not know first name of Banderas.

TableResult

firstname	lastname
George	Clooney
Brad	Pitt
unknown	Banderas

XMLEXISTS

```
SELECT id
FROM EmpXML
WHERE
   XMLEXISTS ( '/emp/date lt "2001-04-23"'
      PASSING BY VALUE EmpXML.ColEmpXML )
```

XPath query

id
1001
1006
…

Database applications

- NDBI026
  - Oracle and MS SQL Server (alternatives)
    - embedded SQL, administration
    - external applications
    - indexing, optimisations
    - transactions
    - security
  - see http://www.ms.mff.cuni.cz/~kopecky/vyuka/dbapl/ http://www.ms.mff.cuni.cz/~kopecky/teaching/dbapps/