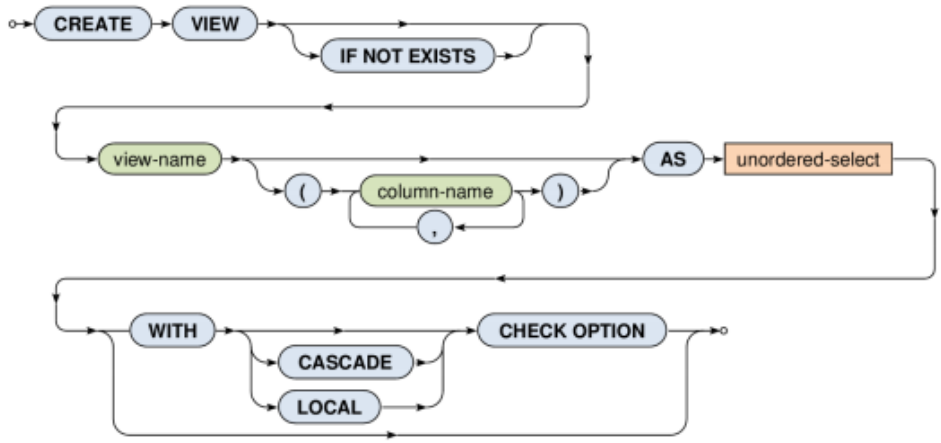
SQL: Advanced Constructs (5)

Views (pohledy)

- pojmenované SELECT dotazy
- vyhodnocované dynamicky
- mohou být použity jako tabulky (např. ve FROM klauzuli)
- dobré pro tvorbu virtuálních tabulek, bezpečnost (schovávání tabulek), opakované používání komplikovaných dotazů,…

CREATE VIEW

- View name and optionally names of its columns
- Select query and check option



- obsah může být **updatován**, ale jen někdy a je-li explicitně povoleno
  - tj. specifikováno WITH CHECK OPTION
  - dává to smysl
    - view založen na jednoduchém SELECT dotazu (bez agregací, subdotazů,…)
    - jen s projekcemi (bez odvozených hodnot) a výběrem z právě jedné tabulky (bez joinů)
    - jsme tedy deterministicky schopni rekonstruovat veškeré tuply ke vložení/aktualizování v originálních tabulkách
  - nově vložené/aktualizované tuply budou viditelné…
    - **LOCAL** – v daném view
    - **CASCADE** (default) – v daném view a ostatních, ze kterých je tento view odvozen

View creation

```
CREATE VIEW BigPlanes AS
SELECT * FROM Aircrafts WHERE (Capacity > 200)
WITH LOCAL CHECK OPTION
```

Successful insertion

```
INSERT INTO BigPlanes
VALUES ('Boeing 737', 'CSA', 201);
```

Denied insertion

```
INSERT INTO BigPlanes
VALUES ('Boeing 727', 'CSA', 100);
```

– This aircraft is only too small (will not be visible in the view)

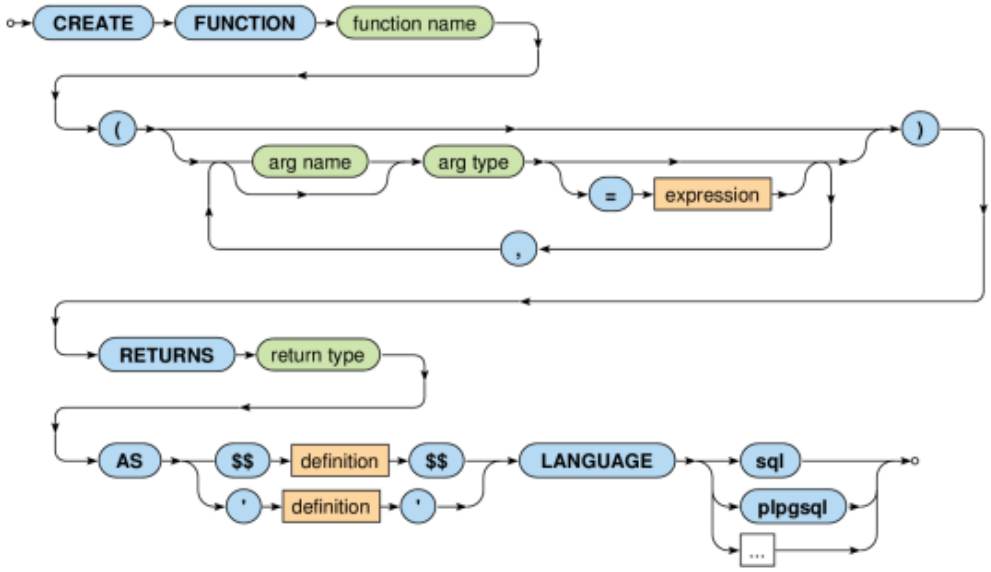
Embedded SQL

- interní databázové aplikace
- proprietární procedurální rozšíření SQL (PL/pgSQL u PostgreSQL)

- dostupné konstrukty:
  - řídící příkazy – if then else, for, while, switch
  - funkce
  - ukazatele – iterativní skenování tabulek
  - triggery – všeobecná integritní omezení

Functions (stored procedures)

CREATE FUNCTION – defines a new function



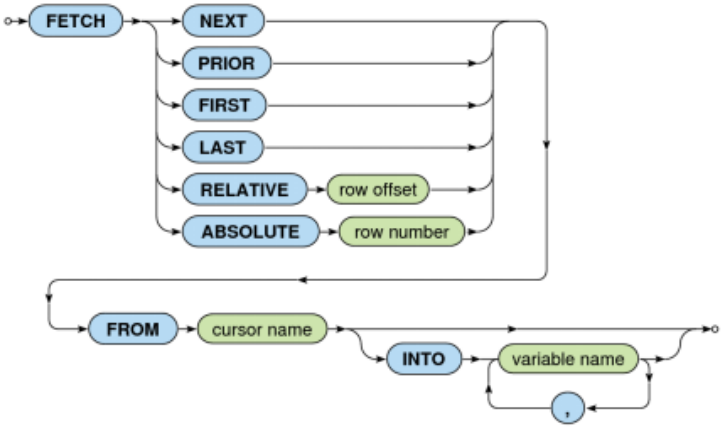

```
CREATE FUNCTION inc(x INT)
RETURNS INT
AS
$$
  BEGIN
    RETURN x + 1;
  END;
$$
LANGUAGE plpgsql;

SELECT inc(5);
```

Cursors (ukazatele)

- databázový ukazatel je řídící struktura, která nám umožní procházet řádky zvolené tabulky

- **SCROLL** – povolí i zpětné fetche

- získání dat:



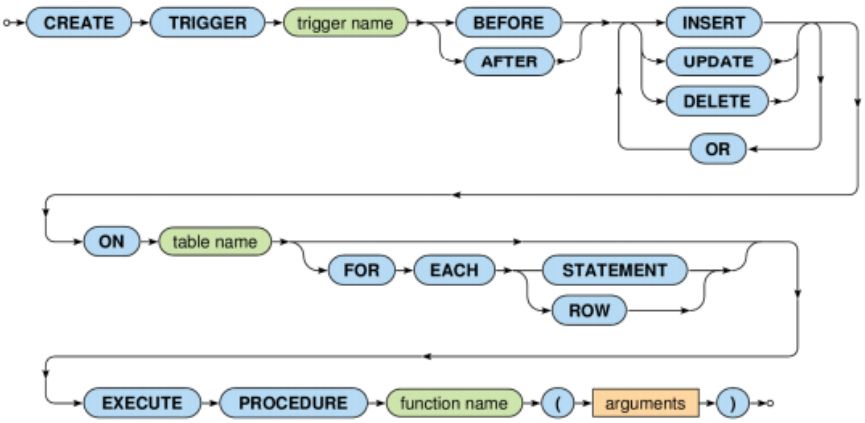
- **INTO …**
  - lokální proměnné, do kterých bude daná řádka uložena
  - nejsou-li další řádky, vrátí NULL

Triggers

- trigger je procedura, která je automaticky vykonána jako odpověď na určité události (INSERT, UPDATE, DELETE)
- používány pro udržování komplexních integritních omezení
- **módy**:
  - **FOR EACH STATEMENT** (default)
    - trigger bude spuštěn jen jednou pro všechny řádky z daného dotazu
  - **FOR EACH ROW**

- **PROCEDURE** – návratový typ musí být TRIGGER

CREATE TRIGGER



SQL/XML

- rozšíření SQL o XML data
- XML datatype, konstrukty (funkce, konstruktory, mapování, XQuery embedding,…)

Table: books

id	catalogue	title	details	language
1	c1	Red	<author>John</author> <author>Peter</author>	en
2	c1	Green	<price>25</price>	NULL
3	c2	Blue	<author>John</author>	en

Table: languages

code	name
en	English
cs	Czech

Query

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

Result

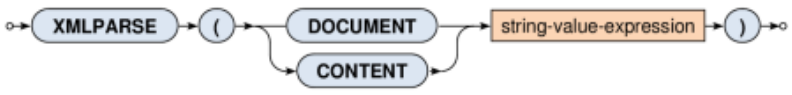
id	
3	<book> <title>Blue</title> <author>John</author> </book>
1	<book> <title>Red</title> <author>John</author> <author>Peter</author> </book>

XML Datatype

- tradiční typy (BLOB, CLOB, VARCHAR,…)
- **nativní XML typ** (kolekce informačních itemů)
  - založen na XML Information Set (XML Infoset)
  - elementy, atributy, procesní instrukce,…
  - povolujeme i fragmenty bez právě jednoho root elementu
    - tzn., že XML hodnoty nemohou být XML dokumenty
  - NULL

Parsování XML hodnot

- **XMLPARSE** – vytvoří XML hodnotu ze stringu
  - **DOCUMENT** – korektní (well-formed) dokument s právě jedním rootem
  - **CONTENT** – korektní fragment

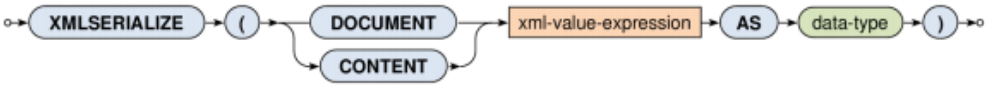



```
SELECT XMLPARSE(
   DOCUMENT "<book><title>Red</title></book>"
) AS result
```

result
<book> <title>Red</title> </book>

Serializace XML hodnot

- **XMLSERIALIZE** – exportuje XMl hodnotu do stringu

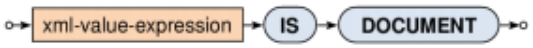



```
SELECT
   id, title,
   XMLSERIALIZE(CONTENT details AS VARCHAR(100)) AS export
FROM books
```

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

Well-formedness predikát

- **IS DOCUMENT** – testuje, je-li XML hodnota XML dokumentem
  - vrátí TRUE, je-li právě jeden root element (jinak FALSE)

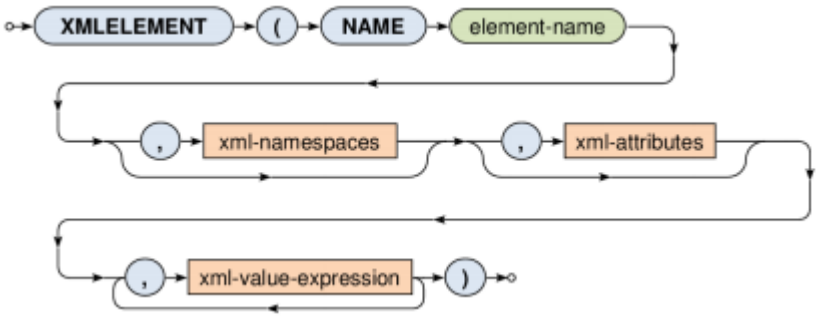


Konstruktory

- funkce pro vytváření XML hodnot
  - **XMLELEMENT** – elementy
  - **XMLNAMESPACES** – deklarace namespace
  - **XMLATTRIBUTES** – atributy
  - **XMLCOMMENT** – komentáře
  - **XMLPI** – procesní instrukce
  - **XMLFOREST** – sekvence elementů
  - **XMLCONCAT** – spojování hodnot
  - **XMLAGG** – agregáty

Elementy

- **XMLELEMENT** – vytvoří XML element s daným jménem a…
  - volitelné deklarace namespace
  - volitelné atributy
  - volitelný obsah elementu



Příklad 1

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

id	result
1	<book>Red</book>
2	<book>Green</book>
3	<book>Blue</book>

Příklad 2: Subelementy

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

id	records
1	<book> <title>Red</title> <language>en<language> </book>
...	...

Příklad 3: Smíšený obsah

```
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>Red</title> with identifier equal to 1. </info>
...	...

Příklad 4: Subdotazy

```
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>Red</title>	<language>**English**</language>
...	...	...

Atributy

- **XMLATTRIBUTES** – vytvoří množinu atributů
  - **INPUT**: seznam hodnot
    - každá hodnota musí mít explicitní/implicitní jméno
      - použito jako jméno pro daný atribut
      - implicitní jména mohou být odvozena (např. ze jmen sloupců)
  - **OUTPUT**: XML hodnota s danými atributy

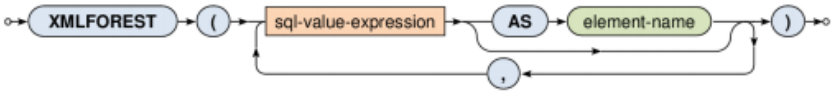
```
SELECT
   id,
   XMLELEMENT(NAME "book",
      XMLATTRIBUTES(
         language, catalogue AS "location"
      ),
      XMLELEMENT(NAME "title", title)
   ) AS book
FROM books
```

id	book
1	<book **language="en" location="c1"**> <title>Red</title> </book>
...	...

Sekvence elementů

- **XMLFOREST** – vytvoří sekvenci XML elementů
  - **INPUT**: seznam SQL hodnot
    - částečné NULL ignorovány, pokud je celý výraz NULL, tak vrátí NULL
    - každá hodnota obsahu musí mít explicitní/implicitní jméno
      - použito jako jméno pro daný element

- **OUTPUT**: XML hodnota s elementy sekvence




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

id	book
1	<title>Red</title> <language>en</language> <location>c1</location>
2	<title>Green</title> <location>c1</location>
...	...

Spojování (Concatenation)

- XMLCONCAT – vytvoří sekvenci z daného seznamu hodnot
  - INPUT: seznam XML hodnot
  - OUTPUT: XML hodnota se sekvencí hodnot


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

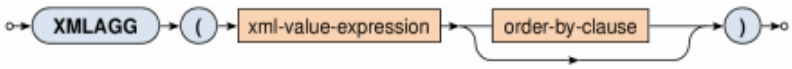
id	description
1	<book>Red</book> <author>John</author> <author>Peter</author>
...	...

XML agregace

- **XMLAGG** – agreguje řádky v rámci dané nadřádky
  - chová se jako standardní agregační funkce (SUM, AVG,…)
  - **INPUT**: řádky z dané nadřádky

- tyto řádky mohou být nejprve volitelně seřazeny (ORDER BY)
- pro každou řádku je vygenerována XML hodnota
- všechny vygenerované XML hodnoty jsou pak spojeny

- **OUTPUT**: XML hodnota se sekvencí itemů




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

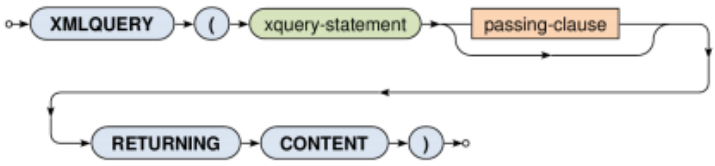
catalogue	list
c1	<book id="1">Red</book> <book id="2">Green</book>
c2	<book id="3">Blue</book>

Dotazování

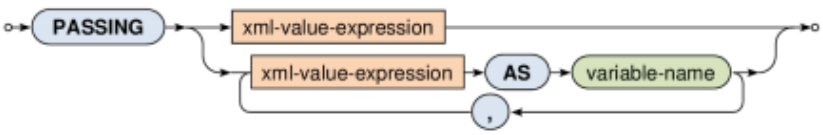
- dotazovací konstrukty, založené na jazyku XQuery
- **XMLQUERY** – vrací výsledek dotazu; většinou v SELECT
- **XMLTABLE** – dekomponuje výsledek dotazu do tabulky; většinou ve FROM
- **XMLEXISTS** – testuje neprázdnost výsledku dotazu; většinou ve WHERE

XMLQUERY

- vyhodnotí XQuery příkaz a vrátí jeho výsledek
- **INPUT**: XML hodnoty deklarované ve volitelné PASSING klauzuli
- **OUTPUT**: XML hodnota



- **vstupní data:**
  - je-li specifikována jen jedna hodnota, její obsah je přístupný přes XQuery příkaz
  - je-li specifikována jedna či více pojmenovaných proměnných, jejich obsah je přístupný přes *$variable-name/*

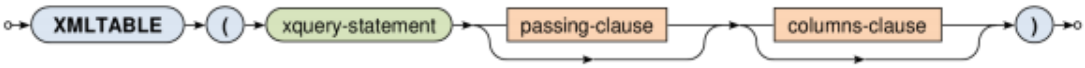


```
SELECT
   id, title,
   XMLQUERY(
      "<authors>{ count($data/author) }</authors>"
      PASSING details AS data
      RETURNING CONTENT
   ) AS description
FROM books
```

id	title	description
1	Red	<authors>2</authors>
...	...	...

XMLTABLE

- dekomponuje výsledek XQuery do virtuální tabulky
- **OUTPUT**:
  - je-li specifikáno COLUMNS, tabulka obsahující výsledek XQuery je rozdělena do jednotlivých řádek a sloupců, a to dle popisu
  - jinak tabulka s jednou řádkou a sloupcem, reprezentující výsledek jako XML hodnotu




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

id	title	result
1	Red	<authors>2</authors>
...	...	...

```
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	Red	2
...	...	...

XMLEXISTS

- testuje neprázdnost výsledku XQuery příkazu
- **OUTPUT**: boolean hodnota (TRUE pro neprázdné sekvence, FALSE pro prázdné)


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

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