1. Przetwarzanie danych XML w bazie PostgreSQL.

Publikowanie (generowanie) dokumentów XML realizowane jest w bazie danych PostgreSQL zgodnie z standardem SQL/XML dostępnym w języku SQL od wersji SQL2003. Standard ten jest wykorzystywany do publikacji dokumentów XML w bazach danych Oracle czy IBM DB2. W bazie MS SQL Server firma Microsoft zaproponowała swoje własne rozwiązanie FOR XML. W ramach technologii SQL/XML użytkownicy relacyjnej bazy danych otrzymują funkcje do tworzenia dokumentu XML na podstawie danych zawartych w relacyjnych tabelach. Baza danych PostgreSQL zaimplementowała większość poleceń standardu SQL/XML.

Funkcja ***xmlelement()***. Umożliwia ono utworzenie węzła - elementu w dokumencie XML na podstawie danych, które nie są typu XML. Składnia funkcji przedstawiona została poniżej.

xmlelement(name name [, xmlattributes(value [AS attname] [, ...])] [, content, ...])

Name - jest nazwą elementu, funkcja xmlattributes() dodaje do elementu atrybuty, kolejne parametry tworzą zawartość elementu. Poniżej przykład utworzenia dokumentu XML zawierającego element FNAME wypełnionego wartością z kolumny FNAME z tabeli sample_table. W kolejnym przykładzie element FNAME zawiera atrybut EMAIL. Dla każdego rekordu tabeli tworzony jest oddzielny dokument XML.

Przykład.

```
SELECT xmlelement ( name  fname, fname) FROM sample_table;

SELECT xmlelement ( name  lname,
            xmlattributes ( email as email ) ,
            lname )
FROM sample_table;
```

Funkcja ***xmlforest()*** umożliwia wygenerowanie poddrzewa XML złożonego z elementów utworzonych na podstawie wyspecyfikowanych nazw atrybutów z tabel relacyjnych.

Przykład.

```
SELECT xmlelement ( name  user,
            xmlattributes ( id as id ),
            xmlforest ( fname as fname, lname as lname, email as email ))
FROM sample_table ;
```

Funkcja ***xmlagg()***, która umożliwia budowanie poddrzewa XML z elementów utworzonych na podstawie wartości pobranych z różnych wierszy tabel relacyjnych.

Przykład.

```
SELECT xmlelement ( name  users,
        ( SELECT xmlagg ( xmlelement ( name lname, lname)) FROM sample_table ) ) ;


SELECT xmlelement ( name  users,
            ( SELECT xmlagg (
                xmlelement ( name  user,
                        xmlattributes ( id as id ),
                        xmlforest ( fname as fname, lname as lname, email as email )))
              FROM sample_table ) ) ;


SELECT xmlroot ( xmlelement ( name  users,
                ( SELECT xmlagg( xmlelement ( name lname, lname)) FROM sample_table ) ),
          version '1.0', standalone yes );
```

Funkcje: ***table_to_xml(), table_to_xmlschema(), table_to_xml_and_xmlschema(), query_to_xml_xmlschema().***

Funkcje powyższe pozwalają odwzorować struktury relacyjne do struktur XML i XML Schema.

Przykład

```
SELECT table_to_xml('sample_table',true,true,'');

SELECT table_to_xmlschema('sample_table',true,true,'');

SELECT table_to_xml_and_xmlschema('sample_table',true,true,'');

SELECT query_to_xml_and_xmlschema('SELECT lname FROM sample_table',true,true,'');
```

2. Typ danych XML w bazie PostgreSQL.

W bazie danych PostgreSQL dostępny jest typ danych XML do przechowywania struktur XML. Do przetwarzania danych zawartych w ramach tego typu udostępniono funkcje oparte o technologię XPath.

Przykład.

```
SELECT xmlelement ( name  user, xmlattributes ( id as id ),
            xmlelement ( name fname, fname ),
            xmlelement ( name lname, lname ),
            xmlelement ( name address, xmlelement ( name city, city )),
            xmlelement ( name contact, xmlforest ( email, phone )) )
FROM sample_table ;


INSERT INTO xml_table SELECT id, xmlelement ( name  user, xmlattributes ( id as id ),
             xmlelement ( name fname, fname ),
             xmlelement ( name lname, lname ),
             xmlelement ( name address, xmlelement ( name city, city )),
             xmlelement ( name contact, xmlforest ( email, phone )) )
             FROM sample_table ;

SELECT id, data FROM xml_table;
```

Funkcje:

xml_is_well_formed(text), xml_is_well_formed_document(text),

xml_is_well_formed_content(text)

pozwalają kontrolować poprawność (czy dokumenty zasługują na określenie "well formed") dokumentów XML.

Funkcje: XMLPARSE ({ DOCUMENT | CONTENT } ***value***),
XMLSERIALIZE ({ DOCUMENT | CONTENT } ***value*** AS ***type***)

pozwalają na konwersję ciągu znaków na typ xml i typu xml na ciąg znaków.

3. Funkcja XPath

Pozwala uzyskać wskazane węzły i rekordy w typie xml.

Przykłady.

```
SELECT id, cast(xpath('//contact/phone/text()', data) as text[]) AS phone FROM xml_table;
SELECT id, xpath('//contact/phone/text()', data)::text AS phone FROM xml_table;
SELECT id, unnest(xpath('//contact/phone/text()', data))::text AS phone FROM xml_table;
```

```
SELECT id, xpath('//lname/text()', data)::text,
       xpath('//address/city/text()', data)::text
FROM xml_table
WHERE cast(xpath('//contact[email="dadacki@google.com"]', data) as text[]) != '{}';

SELECT id, unnest(xpath('//lname/text()', data))::text,
       unnest(xpath('//address/city/text()', data))::text
FROM xml_table
WHERE cast(xpath('//contact[email="dadacki@google.com"]', data) as text[]) != '{}';

SELECT id, unnest(xpath('//lname/text()', data))::text,
       unnest(xpath('//address/city/text()', data))::text
FROM xml_table
WHERE cast(xpath('//contact/email/text()', data) as text[]) = '{dadacki@google.com}';
```

```
SELECT id, xpath('//lname/text()', data)::text,
        xpath('//address/city/text()', data)::text
FROM xml_table
WHERE xpath('//contact/email/text()', data)::text = '{dadacki@google.com}';

SELECT id, xpath_exists('//contact/phone/text()', data) FROM xml_table ;
SELECT id, xpath_exists('//contact/phone/text()', data) FROM xml_table ;
```

4. Typ hstore - dane typu klucz - wartość.

W ramach bazy danych PostgreSQL możliwe jest przetwarzanie wartości klucz - wartość. Dane są przechowywane w atrybucie typu hstore. Do obsługi tego typu danych zostały zaimplementowane odpowiednie funkcje.

Przykłady poleceń SELECT, UPDATE, DELETE:

```
SELECT data->'temp' AS temp FROM hstore_table;

SELECT time, data->'temp' AS temp FROM hstore_table WHERE data->'temp' = '0.0';

UPDATE hstore_table SET data = data || '"prog"=>"v011.2017"' :: hstore;

UPDATE hstore_table SET data = data || '"prog"=>"v012.2017"' :: hstore;

UPDATE hstore_table SET data = delete ( data, 'prog' );

UPDATE hstore_table SET data = data || '"temp0"=>"-1.0"'::hstore WHERE id IN (2,3);
```

Wyszukiwanie wybranych rekordów zawierających odpowiednie klucze w ramach struktury hstore umożliwiają dodatkowe operatory **?**, **@>, <@,?&.**

```
SELECT id, data->'temp0', data FROM hstore_table WHERE data ? 'temp0' ;

SELECT id, time FROM hstore_table WHERE data @> '"temp"=>"0.0"'::hstore ;

SELECT id, time FROM hstore_table WHERE data ?& array [ 'temp','temp0' ] ;
```

Do obsługi typu hstore zostały przygotowane specjalistyczne funkcje.
Funkcje:
akeys() umożliwia wyświetlenie wszystkich kluczy zawartych w danych hstore,
skyes() wyświetla klucze w postaci zbioru rekordów.
Podobnie dla wartości zawartych w strukturze hstore dostępne są dwie funkcje avals() i svals().

```
SELECT akeys(data) FROM hstore_table;

SELECT skeys(data) FROM hstore_table;

SELECT avals(data) FROM hstore_table;

SELECT svals(data) FROM hstore_table;
```

Odnośniki.

8.13. XML Type
9.14. XML Functions
SQL/XML
XML Support
PostgreSQL XML Functions
PostgreSQL XML Type
SUPPORT MULTI-ARGUMENT UNNEST(), AND TABLE() SYNTAX FOR MULTIPLE FUNCTIONS.
F.16. hstore
PostgreSQL hstore