The Road to the XML Type
Current and Future Developments

Nikolay Samokhvalov  Peter Eisentraut

PGCon 2007
Outline

1. Past Developments
2. Current Developments
3. Future Developments
4. Use Cases
5. Conclusion
Outline

1. Past Developments
2. Current Developments
3. Future Developments
4. Use Cases
5. Conclusion
Past Developments

- contrib/xml2 by J. Gray et al.
- Initial patch for SQL/XML publishing functions by Pavel Stehule
- Google Summer of Code 2006 - Nikolay Samokhvalov
- Initial version of export functions by Peter Eisentraut
Outline

1. Past Developments
2. Current Developments
3. Future Developments
4. Use Cases
5. Conclusion
New Features

Target for PostgreSQL 8.3:

- XML Data Type
- XML Publishing
- XML Export
- SQL:2003 conformance
- XPath
Outline

1. Past Developments
2. Current Developments
   - XML Data Type
   - XML Publishing
   - XML Export
   - XPath
3. Future Developments
4. Use Cases
5. Conclusion
CREATE TABLE test (  
    ...,  
    data xml,  
    ...  
);  

Features:  
- Input checking  
- Support functions  

Issues:  
- Internal storage format (plain text)  
- Encoding handling  

Nikolay Samokhvalov, Peter Eisentraut  
The Road to the XML Type
Using the XML Type

Bizarre SQL way:

```sql
INSERT INTO test VALUES (
    ...,
    XMLPARSE (DOCUMENT '<foo>...</foo>'),
    ...)

SELECT XMLSERIALIZE (DOCUMENT data AS varchar)
    FROM test;
```

Simple PostgreSQL way:

```sql
INSERT INTO test VALUES (... , '<foo>...</foo>', ...);

SELECT data FROM test;
```
XML Type Oddities

- No comparison operators
- To retrieve, use:
  - Cast to text, or
  - XPath, or
  - Other key column
- To index, use:
  - Cast to text, or
  - XPath
Producing XML Content

The old way?

```
SELECT '<record id="" || id || '""> <value>
     || ad_hoc_escape_func(value)
   || '</value></record>'
FROM tab;
```

The new way:

```
SELECT XMLELEMENT(NAME record,
   XMLATTRIBUTES(id),
   XMLELEMENT(NAME value, value))
FROM tab;
```
**XML ELEMENT Example**

**SQL:**

```sql
XMLROOT ( 
    XMLELEMENT ( 
        NAME 'gazonk', 
        XMLATTRIBUTES ( 
            'val' AS 'name', 
            1 + 1 AS 'num' 
        ) , 
        XMLELEMENT ( 
            NAME 'qux', 
            'foo' 
        ) , 
        VERSION '1.0', 
        STANDALONE YES 
    ) 
)
```

**Result:**

```xml
<?xml version='1.0' standalone='yes' ?>
<gazonk name='val' num='2'>
    <qux>foo</qux>
</gazonk>
```
XMLFOREST Example

```
SELECT xmlforest (  
    "FirstName" as "FName", "LastName" as "LName",  
    'string' as "str", "Title", "Region" )  
FROM "Demo"."demo"."Employees";
```

might result in

```
<FName>Nancy</FName>
<LName>Davolio</LName>
<str>string</str>
<Title>Sales Representative</Title>
<Region>WA</Region>
...
<FName>Anne</FName>
<LName>Dodsworth</LName>
<str>string</str>
<Title>Sales Representative</Title>
```

(1 row per record)
XMLAGG Example

```sql
SELECT xmlelement ('Emp',
    xmlattributes ('Sales Representative' as "Title"),
    xmlagg (xmlelement ('Name', "FirstName", ' ', "LastName")))
FROM "Demo"."demo"."Employees"
WHERE "Title" = 'Sales Representative';
```

might result in

```xml
<Emp Title="Sales Representative">
  <Name>Nancy Davolio</Name>
  <Name>Janet Leverling</Name>
  <Name>Margaret Peacock</Name>
  <Name>Michael Suyama</Name>
  <Name>Robert King</Name>
  <Name>Anne Dodsworth</Name>
</Emp>
```

(1 row)
Outline

1. Past Developments
2. Current Developments
   - XML Data Type
   - XML Publishing
   - XML Export
   - XPath
3. Future Developments
4. Use Cases
5. Conclusion
XML Export

- Map table/schema/database contents to XML document
- Map table/schema/database schema to XML Schema

Useful for:
- Downstream processing (e.g., SOAP, web services)
- Postprocessing using XSLT
- Backup???
- Display formats (alternative to psql’s HTML mode)
XML Export Functions

Data export:

```sql
table_to_xml(tbl regclass, nulls boolean,
            tableforest boolean, targetns text)
query_to_xml(query text, nulls boolean,
             tableforest boolean, targetns text)
cursor_to_xml(cursor refcursor, count int, nulls boolean,
              tableforest boolean, targetns text)
```

Schema export:

```sql
table_to_xmlschema(tbl regclass, nulls boolean,
                   tableforest boolean, targetns text)
query_to_xmlschema(query text, nulls boolean,
                   tableforest boolean, targetns text)
cursor_to_xmlschema(cursor refcursor, nulls boolean,
                    tableforest boolean, targetns text)
```
XML Schema Mapping Example

```
CREATE TABLE test (a int PRIMARY KEY, b varchar(200));
```

is mapped to

```
<xsd:complexType name="RowType.catalog.schema.test">
  <xsd:sequence>
    <xsd:element name="a" type="INTEGER"></xsd:element>
    <xsd:element name="b" type="VARCHAR_200_200" minOccurs="0"></xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TableType.catalog.schema.test">
  <xsd:sequence>
    <xsd:element name="row"
      type="RowType.catalog.schema.test"
      minOccurs="0"
      maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
```
<catalogname>
  <schemaname>
    <tablename>
      <row>
        <colname1>value</colname1>
        <colname2 xsi:nil='true'/>
        ...
      </row>
      ...
    </tablename>
    ...
  </schemaname>
  ...
</catalogname>
Outline

1. Past Developments
2. Current Developments
   - XML Data Type
   - XML Publishing
   - XML Export
   - XPath
3. Future Developments
4. Use Cases
5. Conclusion
The table for further examples:

```sql
CREATE TABLE table1(
    id INTEGER PRIMARY KEY,
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    xdata XML
);
```
XPath Example

Sample data:

```sql
INSERT INTO table1(id, xdata)
VALUES(1,
  '<dept xmlns:smpl="http://example.com" smpl:did="DPT011-IT">
    <name>IT</name>
    <persons>
      <person smpl:pid="111">
        <name>John Smith</name>
        <age>24</age>
      </person>
      <person smpl:pid="112">
        <name>Michael Black</name>
        <age>28</age>
      </person>
    </persons>
  </dept>
);
```
XPath Example

Simple example:

```
SELECT *
FROM table1
WHERE (xpath('//person/name/text()',
       xdata))[1]::text = 'John Smith';
```

And using namespaces:

```
xmltest=# SELECT *
FROM table1
WHERE (xpath('//person/@smpl:pid', xdata,
       ARRAY[ARRAY['smpl', 'http://example.com']]))::text = '111'
FROM table1;
```
Use functional indexes to avoid XPath evaluation at runtime:

CREATE INDEX i_table1_xdata ON table1 USING btree(
    xpath('//person/@name', xdata)
);

Nikolay Samokhvalov, Peter Eisentraut
The Road to the XML Type
External Dependencies

- Uses libxml (MIT License) for XML publishing and XPath
- Enable with `configure --with-libxml`
- Not necessary for XML export
Future Developments

- DTD and XML Schema validation
- Annotated schema decomposition
- XSLT
- Performance issues
- Full-Text Search
- Advanced Indexing (XLABEL)
- More, More, More
DTD and XML Schema validation

DTD validation:
- Implemented for 8.3, DTD is passed by URI
- Should be extended to allow passing DTD as text

XML Schema (XSD) validation (XMLVALIDATE per SQL:2006):

```sql
INSERT INTO messages(msg)
SELECT xmlvalidate(
    DOCUMENT '<?xml ...
    ACCORDING TO XMLSCHEMA NO NAMESPACE
    LOCATION 'http://mycompany.com/msg-schema'
);
```

*The result of XMLVALIDATE is new XML value!*
Annotated schema decomposition

In some cases decomposition is better (no needs in storing XML data, XML serves as transport only):

- When we need to store only small parts of the XML data
- Already developed tools might be designed only for relational data

During decomposition following capabilities could be used:

- Data normalization
- Foreign keys creation
- Conditional insertion of data chunks
- Insert parts of initial XML document as XML values
The easiest way: adapt and expand contrib/xml2’s capabilities. We should choose one of two:

- Move XSLT functionality to the core (and use --with-libxslt)
- Separate contrib/xslt
Performance issues

Ideas:

- Cache intermediate results to avoid redundant parsing and XPath evaluation
- Advanced physical storage to speedup access to arbitrary node in XML data
- Use PostgreSQL existing capabilities for full-text search
- Use additional structures/tables/indexes to avoid XPath evaluation at runtime
- Use slices (similar to `array_extract_slice()`) to avoid dealing with entire values (both in SELECTs and UPDATEs)
Simple way to create FTS index (available in 8.3):

CREATE INDEX i_table1_fts ON table1
USING gist(
    to_tsvector(
        'default',
        array_to_string(xpath('//@text()', xdata), ' ')
    )
);

Nikolay Samokhvalov, Peter Eisentraut
The Road to the XML Type
Proposal for overloading of built-in `to_tsvector()`:

```sql
CREATE OR REPLACE FUNCTION to_tsvector(text, xml)
RETURNS tsearch2.tsvector
AS $BODY$
    SELECT to_tsvector($1, array_to_string(xpath('//text()', $2), ' '));
$BODY$ LANGUAGE sql IMMUTABLE;

CREATE INDEX i_table1_fts
ON table1
USING gist(to_tsvector('default', xdata));
```
Further ideas for full-text search:

- Indexing parts of documents (available in 8.3, in some way)
- Element names in `tsvector`
- Relevance Scoring (ranking)
- FTS parser for XML
Idea:

- Enumerate all XML node names in one database-wide table \((xnames)\)
- Store shredded data in additional table \((\text{columnname}_x\text{label})\)
- Use numbering scheme (in prototype it’s \(\text{ltree}\), then SLS) to encode nodes
- Use GiST/GIN indexes for numbering scheme column
- Rewrite XPath expression to plain SQL statement
- Implement partial updates support to avoid massive index rebuilding
Enumerate all XML node names in the database:

**Table: xnames**

<table>
<thead>
<tr>
<th>xname_id</th>
<th>xname_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>person</td>
</tr>
<tr>
<td>2</td>
<td>dept</td>
</tr>
<tr>
<td>3</td>
<td>name</td>
</tr>
<tr>
<td>4</td>
<td>did</td>
</tr>
<tr>
<td>5</td>
<td>persons</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
For an XML column implicitly create additional table (using `xlabel.register_column()` function):

**Table:** table1_xdata

<table>
<thead>
<tr>
<th>tid</th>
<th>xlabel</th>
<th>node_type</th>
<th>xname_id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>1 (elem.)</td>
<td>2</td>
<td>NULL</td>
</tr>
<tr>
<td>1</td>
<td>a.b</td>
<td>2 (attr.)</td>
<td>4</td>
<td>DPT011-IT</td>
</tr>
<tr>
<td>1</td>
<td>a.c</td>
<td>1 (elem.)</td>
<td>3</td>
<td>NULL</td>
</tr>
<tr>
<td>1</td>
<td>a.c.a</td>
<td>NULL</td>
<td>NULL</td>
<td>IT</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>a.d.a.b</td>
<td>1 (elem.)</td>
<td>3</td>
<td>NULL</td>
</tr>
<tr>
<td>1</td>
<td>a.d.a.b.a</td>
<td>NULL</td>
<td>NULL</td>
<td>John Smith</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

CREATE INDEX i_table1_xdata_xlabel
ON table1_xdata
USING gist(xlabel);
Rewrite XPath expression to plain SQL statement:

```sql
SELECT *  
FROM table1  
WHERE array_dims(xpath('//person/name', xdata)) IS NOT NULL;
```

... becomes ...

```sql
SELECT *  
FROM table1  
WHERE EXISTS(
    SELECT 1  
    FROM table1_xdata AS t1, table1_xdata AS t2  
    WHERE t1.xname_id = 1 AND t2.xname_id = 3  
    AND t3.xlabel <@ t1.xlabel
);
```

... where `<@` means “is a child of”
Current thoughts:

- Separate table is not good (*deja vu*: fti VS tsearch2)
- It would be great if one structure solves 2 problems at once:
  - access to arbitrary node
  - SELECTs with XPath
More, more, more

- Inline ORDER BY for XMLAGG (SQL:2003)
  
  ... XMLAGG(XMLELEMENT(...) ORDER BY coll) ...

- XMLCAST (SQL:2006)

- XML Canonical

- Pretty-printing XML

- Registered XML Schemas (SQL:2006)

- Schema evolution

- Improve Data Model (XDM)

- XQuery Support (SQL:2006)

- Updatable XML views (over relational data)

- RelaxNG validation
And even more!

- Bulk loader for XML data (parallelize the XML parsing)
- XML-awareness in APIs and PLs
- Additional contribs/projects (web services, ODF, DocBook utils, etc)
- New tools and applications, integration with existing ones
Outline

1. Past Developments
2. Current Developments
3. Future Developments
4. Use Cases
5. Conclusion
Use Cases

- Use Case 1: Document Management System
- Use Case 2: Store Logs in the Database
- Use Case 3: Heterogeneous Catalog
Use Case 1: Document Management System

The primary goal: to store documents in the RDBMS *as is*
Use Case 2: Store Logs in the Database

Table: action

<table>
<thead>
<tr>
<th>action_id</th>
<th>SERIAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>action_type_id</td>
<td>INT4</td>
</tr>
<tr>
<td>action_status_id</td>
<td>INT4</td>
</tr>
<tr>
<td>action_person_id</td>
<td>INT4</td>
</tr>
<tr>
<td>action_data</td>
<td>XML</td>
</tr>
</tbody>
</table>

The primary goal: to achieve flexibility, avoid DB schema changes (schema evolution)
Use Case 3: Heterogeneous Catalog

Task: to build heterogeneous catalog (items of different types, a lot of properties)
Use Case 3: Heterogeneous Catalog

Task: to build heterogeneous catalog (items of different types, a lot of properties)

How?
Use Case 3: Heterogeneous Catalog

Ugly way

<table>
<thead>
<tr>
<th>item</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>obj_id</td>
<td>INT8</td>
</tr>
<tr>
<td>item_section_id</td>
<td>INT8</td>
</tr>
<tr>
<td>item_vendor_id</td>
<td>INT8</td>
</tr>
<tr>
<td>item_model_id</td>
<td>INT8</td>
</tr>
<tr>
<td>item_year</td>
<td>INT2</td>
</tr>
<tr>
<td>item_price</td>
<td>NUMERIC(30,6)</td>
</tr>
<tr>
<td>item_prop1</td>
<td>INT4</td>
</tr>
<tr>
<td>item_prop2</td>
<td>INT4</td>
</tr>
<tr>
<td>item_prop3</td>
<td>INT4</td>
</tr>
<tr>
<td>item_prop4</td>
<td>INT4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>item_prop21</td>
<td>TEXT</td>
</tr>
<tr>
<td>item_prop22</td>
<td>TEXT</td>
</tr>
<tr>
<td>item_prop23</td>
<td>TEXT</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>item_prop41</td>
<td>BOOLEAN</td>
</tr>
</tbody>
</table>
Use Case 3: Heterogeneous Catalog

Entity-Attribute-Value model

```
<table>
<thead>
<tr>
<th>item</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>obj_id</td>
<td>INT8</td>
</tr>
<tr>
<td>item_section_id</td>
<td>INT8</td>
</tr>
<tr>
<td>item_vendor_id</td>
<td>INT8</td>
</tr>
<tr>
<td>item_model_id</td>
<td>INT8</td>
</tr>
<tr>
<td>item_year</td>
<td>INT2</td>
</tr>
<tr>
<td>item_price</td>
<td>NUMERIC(30,6)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>item_props</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ip_item_obj_id</td>
<td>INT8</td>
</tr>
<tr>
<td>ip_id</td>
<td>INT4</td>
</tr>
<tr>
<td>ip_value</td>
<td>TEXT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dictionary</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary_id</td>
<td>INT4</td>
</tr>
<tr>
<td>dictionary_key</td>
<td>VARCHAR(32)</td>
</tr>
</tbody>
</table>
```
Use Case 3: Heterogeneous Catalog

Semi-structured data approach

<table>
<thead>
<tr>
<th>item</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>obj_id</td>
<td>INT8</td>
</tr>
<tr>
<td>item_section_id</td>
<td>INT8</td>
</tr>
<tr>
<td>item_vendor_id</td>
<td>INT8</td>
</tr>
<tr>
<td>item_model_id</td>
<td>INT8</td>
</tr>
<tr>
<td>item_year</td>
<td>INT2</td>
</tr>
<tr>
<td>item_price</td>
<td>NUMERIC(30, 6)</td>
</tr>
<tr>
<td>item_props</td>
<td>XML</td>
</tr>
</tbody>
</table>
Use Case 3: Heterogeneous Catalog

Metadata Query Interface for Heterogeneous Data Archives (International Virtual Observatory): http://alcor.sao.ru/php/search/
More Information

  http://wiscorp.com/sql200n.zip

- PostgreSQL documentation.  
  http://momjian.us/main/writings/pgsql/sgml/

- XML Development Wiki Page.  

  http://samokhvalov.com/syrcodis2007.ps