Adding Email Type into a Commercial Object-Oriented Relational Database – TOR

Hongfei Guo : guo@cs.wisc.edu
Wanli Yang: ywl@cs.wisc.ecu

Abstract

One of the key object-relational features added to relational database is an extensible type system (ADTs). In this project, we examined in depth how a real ORDBMS supports this feature by adding a useful new data type – Email ADT into a commercial Object-Oriented Relational Database – NCR's Teradata ORDBMS (TOR). This work involves two parts. Part 1, implementing Email ADT, which consists of a source file, a header file and a XML description file, supporting users to randomly retrieve different parts from Email data. Part 2, integrating Email ADT into TOR, which includes registering Email type, registering user visible functions to operate Email type data, adding rules in parser, and updating build script. Finally, we gave out our test case for Email ADT in addition to the original test case suites for existing data types in TOR and corresponding results.

1 Introduction

One of the key object-relational features added to relational database is an extensible type system (ADTs). It is a striking advance in RDBMS functionality as the needs to manage new data types in the real world never cease to grow since first relational database systems were first introduced approximately 20 years ago. In fact, emerging SQL3 standards today already include many non-traditional data types such as text, video, audio, image, and geo-spatial object data, which are intensively used by a broad range of applications in industries, yet not well served by types pre-built in RDBMS. Examples of possible applications of non-traditional data include medical industries, such as medical training, synthetic interviews, and patient diagnosis; E-commerce industries, such as product information database, webstore, and product and customer information retrieval;

and entertainment industry, such as information repository, location finder and theatre locator; and more.

In this project, we examined in depth how a real ORDBMS supports this feature by adding a useful new data type – Email ADT into a commercial Object-Oriented Relational Database – NCR's Teradata ORDBMS (TOR).

1.1 TOR Features and System Architecture

At NCR, the key element in the shift in paradigm from purely relational to object – relational is the development of new technology that supports traditional as well as non-traditional data types. The result of the development effort is the Teradata Object-Relational Database management System (OEDBMS). Teradata ORDBMS combines attributes inherent in the relational product with capabilities based on emerging standards, including:

- Support of emerging SQL3 standards for non-traditional data types, such as text, video, audio, image, and geo-spatial object data
- Architecture designed from the beginning to support parallel operations
- Massively parallel architecture designed to support high-end databases

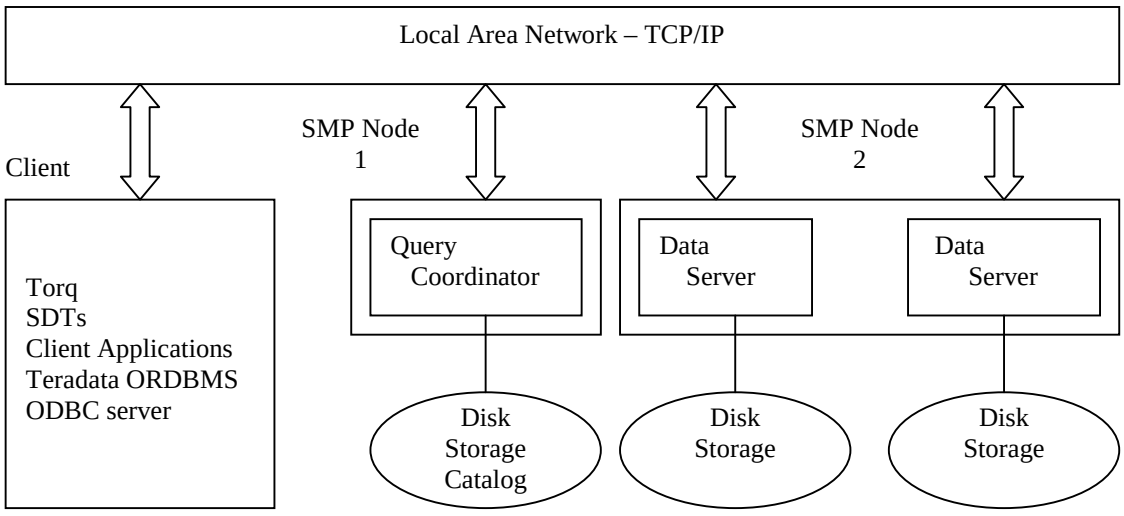

Fig 1-1 General Configuration of TOR

Teradata ORDBMS is a group of client and data server software components. Figure 1-1 represents an MPP constructed when two SMP nodes are connected through a network to form a larger configuration. The Teradata ORDBMS client consists of several software programs described below.

- Teradata ORDBMS ODBC Driver based on the Microsoft ODBC API. The driver with extensions provides a Call Level Interface (CLI) that client applications written in C++ use to access the Teradata ORDBMS SDT library.
- TOR Query Facility: Torq provides a user interface to the Teradata ORDBMS.
- TOR SQL: used to formulate queries for information involving STDs. To support the new STDs, TOR SQL enhances the existing Teradata SQL with Data Definition Language (DDL) extensions based on SQLs.
- STD Library: Contains a set of pre-defined STDs, which is installed as part of the client software on a PC. Through the library, an application has access to the methods and functions in the STDs.

The Teradata ORDBMS server consists of several software programs:

- Query Coordinator (QC): Parses and optimizes a query received from a client application, then creates a plan that it uses to generate instructions for each data server. Teradata ORDBMS contains a single QC;
- Data server: The data server receives a plan from the QC and carries out the instructions. Teradata ORDBMS supports a maximum of 16 data servers distributed among 4 nodes.
- System catalog: System catalogs are a set of tables that store information about all the tables and functions in all the databases that make up a Teradata ORDBMS. Included in the catalogs is information about:
- Utilities: A number of utilities included in Teradata ORDBMS support installation, configuration and interaction with Teradata ORDBMS.

The outline of the rest of this paper is as follows: In section 2, we examine data types in TOR and general rules for adding a new ADT into this system. In section 3, we describe RFC822 email format standards we support, and user interfaces we provide to operate

Email type data. Then we discuss our implementation of Email type and the way to "hook" it up to TOR system in section 4 and section 5. In the next section, we give out our evaluation of the updated system based on our test cases and corresponding results. Finally, we conclude in section7.

2 Data Type in TOR

2.1 What a Data Type Means to TOR

In TOR, a data type is composed of an implementation, a description, and some registration:

- Implementation: A C++ class that implements the data storage and operations supported by the type, include several virtual functions that must be implemented for each type, and functions specific to the type.
- Description: An XML file that specifies the capabilities of the functions, such as member functions available, comparison operations, and indexability, etc, which is used to generate some of the registration information for the type.
- Registration: The modification of several certain existing source files in order to bootstrap the registration of the new type, and make it known to TOR.

After a type is implemented, and the registration code is in place, the type will be available to be used from SQL for defining tables, and for use in queries. Besides, any member functions that have been defined and registered will be available to be called from SQL as well.

2.2 Existing Data Types in TOR

In Teradata ORDBMS, all data are instances of System Defined Data Types (STDs), including traditional and non-traditional data types. A set of predefined STDs are included as part of Teradata ORDBMS program software. Existing STDs are categorized as:

- General, includes: TINYINT, SMALLINT, INT, BIGINT, DECIMAL, REAL, BOOL, DATE, TIME, TIMESTAMP, BLOB(binary large object);
- Multimedia, includes: VIDEO, RASTER8(8-bit deep bitmap), RASTER16(16-bit deep bitmap), RASTER24(24-bit deep bitmap), GIF, AUDIO, DICOM3, and VIDIO1 and VIDIO2(place holder for video data);
- Geo-spatial, includes: RECTAGLE, POINT, CLOSEDPOLYGON, POLYLINE, CIRCLE, and SWCHEESEPOLY(loosely connected polygonal region);
- Array, includes: ARAYUNIT1(multidimensional array of single-byte signed integers), ONEDARRAYINT4(Single dimensional array of 4-byte signed integers), ARRAYUNIT2(Multidimensional array of two-byte unsigned integers), and ONEDARRAYCHAR(Single dimensional array of characters and signed 4-byte integers), and ARRAYINT4(multidimensional array of unsigned 4-byteintegers);
- Text, includes: STRING, TEXT, CLOB(Variable length large string of single byte characters), and PDF

The source files and XML description files of each ADT are organized into its corresponding category directory.

3 Design of Email ADT

In the age of Internet, email is massively used for communication. Providing Email ADT data storing and structured info (for example, subject, sender, etc) retrieving will extend the range of possible applications of an ORDBMS in industries.

3.1 RFC 822 Standard Supported by Email ADT

In Email ADT, we support email standard specified in RFC822:

```
message = fields * (CRLF * text)
// everything after the first null line is message body
field = field-name ":" [field-body] CRLF
field-name = 1* <any CHAR, excluding CTLs, SPACE, and ":">
text = <any CHAR, including bare CR & bare LF, but NOT including CRLF>
```

An email message consists of header fields and, optionally, a body. The body is simply a sequence of lines containing ASCII characters, which is separated from the headers by a null line. Each header field can be viewed as a single, logical line of ASCII characters, comprising a field-name and a field-body. Different parts of the body are also separated by CRLF. See figure 3-1.

Date: …	**\r\n**
From: …	**\r\n**
To: …	**\r\n**
Subject: …	**\r\n**
Other headers	**\r\n**
NULL Line	**\r\n**
Part 1	**\r\n**
Part 2	**\r\n**
Part 3	**\r\n**

Fig 3-1 RFC 822 Standard Email Format

3.2 User Interfaces for Operating Email ADT

Our Email ADT provides means to store an entire MIME encoded email package and to randomly retrieve different parts and headers of it from database. Currently, the following functions are supported for SQL:

`email_num_parts( )`

This function returns the number of parts in a multi-part document. Header is not counted as a part. So the numberof parts may be 0 if the mail contains only headers.

`email_str_part( int i)`

This function returns the specified document part as a string. Currently, the length of the return string is limited to 2048 bytes.

`email_num_headers( )`

This function returns the number of headers in the email-message.

`email_subject( )`

This function returns the subject of the email-message as a string.

`email_from( )`

This function returns the sender of the email-message as a string.

`email_to( )`

This function returns the addressee of the email-message as a string.

`email_date( )`

This function returns the date of the email-message as a string.

4 Implementation of Email ADT

Our implementation of Email ADT can be divided into two parts. Part one is the implementation of general methods – virtual methods needed to be implemented by every ADT, providing the way to create and store Email type object. Part two is the implementation of those methods specially provided by Email ADT for users to manipulate Email type data. Following, we describe these two parts respectively.

4.1 Implementation of general methods

EmailAdt class is inherited from OneDArrayAdt<char> class, which is a standard ADT in TOR for handling "really large" items for a single object. It simplified our work on storing and retrieving Email object to and from database. Figure 4-1 shows the inheritance hierarchy involving EmailAdt:

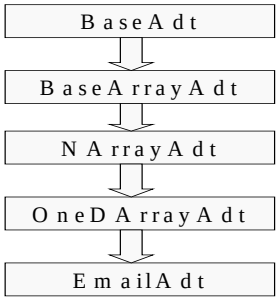

Fig 4-1 Inheritance Hierachy

In order to get high performance when retrieving information from Email ADT data, we parse the data once it is loaded into database, and store the parsing results into two buffers (i.e. desc[] and counts[]) kept in its corresponding emailAdt type object. In this

way, once queried, information can be retrieved directly from buffers instead of by parsing the whole email again and again.

```
enum emailStringTypes {date, header_from, header_to, subject,
                       part1, part2, part3, part4, part5,
                       emailNumStrings};
enum emailInt4Types {headers, parts, emailNumInts};

StringAdt desc[emailNumStrings];  // stores descriptor strings
Int4Adt counts[emailNumInts];     // stores statistics info
```

The workflow of Email ADT is as follows:

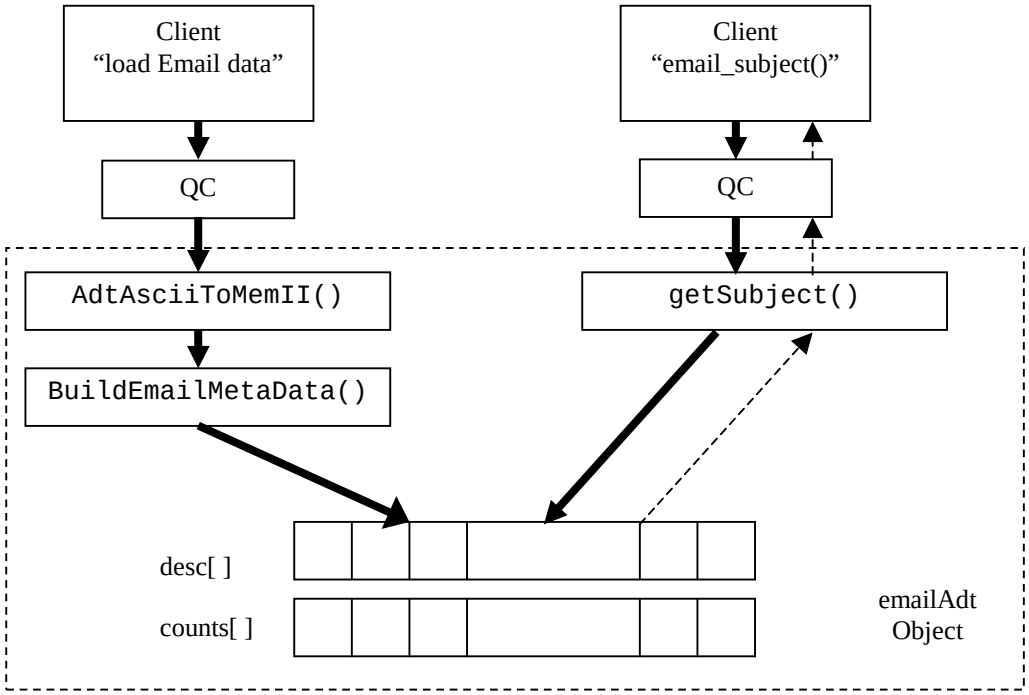

Fig 4-2 Work Flow od Email ADT

When user input a query loading email data, QC(Query Coordinator) will parse the query, and invoke AdtAsciiToMemII() to load Email data into database and create an emailAdtobject for it. Then BuildEmailMetaData() is called in AdtAsciiToMemII() to read this email in from database and parse it. Infomation abstracted is stored into desc[] and counts[]. After that, whenever the user calls a SQL function, QC will simply return required data from those two buffers.

Following, we give out the pseudo code for the email parser BuildEmailMetaData():

```
int EmailAdt::BuildEmailMetaData()
```

```
{

    read the header of the document into buffer;
    if (it is not in email format)
        return error;

    if (the first header is null)
        return error;

    // parese headers
    find Date, put it into desc[date]; headerNo ++;
    find From, put it into desc[header_from]; headerNo ++;
    find To, put it into desc[header_to]; headerNo ++;
    find Subject, put it into desc[subject]; headerNo ++;

    // parse parts
    partNo = 0;
    read a block of the document into buffer;
    while (buffer not empty) {
         scan buffer {
            find part seperator, put this part into desc[partNo];
            partNo++;
         }
         read a block of the document into buffer;
    }
    put headerNo into counts[headers];
    put partNo into counts[parts];

} //BuildEmailMetaData
```

Since an email message might be very large, we cannot read the whole message into memory and parse it. To solve this problem, we read in a block of data once a time and parse it.

Currently, for each part in the email body, the maximum length can be retrieved is the buffer size, since the length of a String type is limited. However, a client wrapper class of emailAdt can be added to the system in order to dump the whole email if it is needed in the future.

4.2 Implementation of special methods

Following are special methods provided by emailAdt, each of which is implementation of corresponding Email ADT SQL function:

```
int getNumParts(Int4Adt& ret) const
```

```
int getStrPart (const Int4Adt& i, StringAdt& ret) const
int getNumHeaders(Int4Adt& ret) const
int getSubject(StringAdt& ret) const
int getFrom(StringAdt& ret) const
int getTo(StringAdt& ret) const
int getDate(StringAdt& ret) const
```

As discussed in section 4.1, these functions simply return expected info from buffer desc[] or counts[].

5 Integrating Email ADT into TOR

Generally, there are several steps to add a new ADT type into TOR:

- Make this ADT type visible to system;
- Register user visible functions to system;
- Add information about this ADT in respect of sorting, indexing, or partition if supported;
- Define selecting and dumpling on this ADT if expected;
- Add rules regarding this ADT into parser;
- Update build script.

Table 5-1 represents basic classes in TOR dealing with all ADT types.

Class	**Description**
AdtInfo	Contains information about all adts registered in the system. There is a single instance of this class in the running TOR system. It is used to dispatch member functions, and create new instances of ADTs when necessary. The AdtInfo object knows about all types and how to create instances, and call their member functions.
BaseAdt	The base class for all ADTs. baseAdt implements a number of functions that are needed by all ADTs. There are a few virtual functions (comparison, for example) that must be implemented by an ADT, which provide a way to perform common operations on values within the engine without having to know the type of the value.
BaseAdtInfo	This class is the base class for AdtInfo and contains the enumeration listing all types defined in the system.

Table 5-1 Basic Classes in TOR involving ADT

Table5-2 represents files involving ADT registration in TOR:

File	Description
adt/base/all_adt.cpp	Contains definition for parts of the AdtInfo class
adt/base/adtinfo.h	Contains the declaration for the AdtInfo class.
BaseAdt.h	Contains the definition for the AdtInfo class. Also contains the definition for the base class for all ADTs: baseAdt. The AdtInfo::initTypes() function initializes several arrays that contain information about types that are based on (subclasses of) the array type.
adt/base/adtMacros.h	Contains definitions for several utility macros used in the implementation of ADTs.
adt/base/torreg.dtd	Contains the XML document type definition for the XML files that describe data types.
adt/base/adtTPL.cpp	Contains the definition for the type precedence list. This list determines what conversions are possible for a given type.
adt/common/baseAdt Info.h	Contains the enumeration for all types.

Table 5-2 Files in TOR involving ADT Registration

Following, we illustrate how these steps got done in detail by registering Email ADT into this system.

5.1 Making Email ADT Visible to TOR

- Edit the "tools/GenAdtFuncReg.pl" to add mappings for the client and server side ADT class names.
- Add Email ADT to the enumeration in "src/adt/common/baseAdtInfo.h".
- Add an entry for Email ADT to the "ptk/misc.tcl"
- Add "#include emalAdt.h " to "src/adt/base/all_adt.h".
- Three changes need to be made in "src/adt/base/all_adt.cpp". First, add a prototype adt instance to the all_adt array. Second, add a line to copy "Email" into the adtName[] array. Third, add an entry to the _sizes[] indicating the number of bytes used by Email ADT is variable length.

- Add a case to the type creation switch statement in "src/client-api/cplusplus/clientAdtInfo.cpp" so that Email ADT can be instantiated on a client.

5.2 Register User Visible Functions to TOR

- Add all functions, including the constructor to the eFuncId enumeration in "src/adt/base/adtInfo.h".
- Add a case to the AdtInfo::invokeConstructor() function to handle the constructor calls in "src/adt/base/all_adt.cpp".
- To be able to call the methods we have defined for Email ADT, we define the precedence list for Email ADT in "base/adtTPL.cpp". This is the precedence for automatic type conversions.

5.3 Adding Rules in the Parser

- Update the parser in src/parse_opt/query/Bparser.y, adding a token definition for Email ADT. Add rules in the LargeObjectTypeName and Ident productions to recognize it.
- Add the type to the keyword table in src/parse_opt/query/pqlexkey.cpp. Note: this table must be put in alphabetic order.

5.4 Update Build Script

- Update src/catalog/GenCatalog.pl to list the new type. A section of this perl script contains a line that calls AddBaseType() for each data type
- Add a gen_adt_function_registrations(libadt_standard, Demo) line to the Imakefile in Email ADT's directory – src/adt/text/.
- Add emailAdt.h to the list of dependencies in the Imakefile in its directory.

- Add emailAdt.cpp to the list of dependencies for the library in the same Imakefile.

6 Evaluation

After adding Email ADT to TOR, we tested this system in two steps. First, we used original TOR ADT test suites to check if the modified system works for originally existing ADTs in TOR. The system passed all these tests.

Then we focused on Email ADT, examining if it works in different kinds of queries. Our test cases mainly include: load table from files; use each Email ADT specific function; apply equality and inequality comparison on Email type attributes; apply equality and inequality comparison between an Email type attribute and a file; apply join on Email type attributes; update Email attribute; insert and delete rows based on Email type attribute etc.. In addition, we included negative cases in our tests such as to load an Email from a file in illegal email format. In order to cover common cases, we also chose different email messages in different formats. For example, email with multi receivers, email without subject, email with word document as an attachment, email with plain text file as an attachment, and emails with multiple parts etc.. The system passed all these test cases as well. (See sample test cases and corresponding results in appendix.)

7 Conclusion

In this project, we examined in depth how a real ORDBMS supports an extensible type system (ADTs) by adding a useful new data type – Email ADT into a commercial Object-Oriented Relational Database – NCR's Teradata ORDBMS (TOR). In our implementation, we support randomly retrieving different parts of Email ADT data in RFC822 standard format by using SQL function call. Then we integrated Email ADT into TOR by registering Email type into system, registering user visible functions, adding rules in parser, and updating build script. Finally, based on our test results for Email ADT as well as test results of original ADT test case suites in TOR, we can say that updated TOR still work fine with original data types and that Email ADT has already been implemented and integrated into TOR successfully.

Reference

1. *Introduction to Teradata ORDBMS.* NCR document B035-1077-099K.
2. Michael J. Carey, David J.Dewitt, Jeffrey F.Naughton. *The BUCKY Object-Relational Benchmark*. In SIGMOD 97.
3. *How to Add a New Data Type to TOR*. NCR document.
4. *TORQ Utility*. NCR document.
5. Stonebraker. *Inclusion of New Types in Relational Database Systems*.