SQL/MED and More

Management of External Data in PostgreSQL and Microsoft SQL Server

Seminar Database Systems

Master of Science in Engineering
Major Software and Systems
HSR Hochschule für Technik Rapperswil
www.hsr.ch/mse

Supervisor: Prof. Stefan Keller
Author: Florian Schwendener
Rapperswil, December 2011

Abstract

Connecting different Database Management Systems (DBMS) always is a hard task. The only way is to learn the multiple technologies like Microsoft Linked Servers (MSSQL Server), DBLink (Oracle) or any other technology implemented in the diverse available products. To solve this issue, a new standard called *SQL/MED* – which stands for Management of External Data – was born. It contains two new concepts: *Foreign Data Wrappers* and *Datalinks*.

This paper sheds light on this rather unknown young technology and examines carefully what it has to offer and how the insides work. In order to appreciate the capabilities of SQL/MED, we show the similarities with Linked Servers by Microsoft and point out how they differ from each other. This happens with the aid of a practical example in which a connection – in both directions – is established between Microsoft SQL Server 2008 and PostgreSQL 9.1 (with support by a *Foreign Data Wrapper*). Having no available implementation of *Datalinks*, this concept is only addressed on a theoretical base.

To sum up the paper, a few advanced topics related to *SQL/MED* and PostgreSQL are presented. This includes two interesting *Foreign Data Wrappers* and a detailed look on some features of SQL/MED. A conclusion tries to predict the near future of this very interesting technology.

Keywords

SQL/MED, Foreign Data Wrapper, Datalinks, PostgreSQL, external data, Microsoft SQL Server, Linked Servers, OLE DB, ODBC

Table of Contents

1 Introduction

Both PostgreSQL (currently in version 9.1) and Microsoft SQL Server (2008 R2) are widely used database servers. While PostgreSQL is open source and controlled by the community, MSSQL is closed source and is only commercially available. Both are full-featured and also share their first release year (which was 1989). Microsoft started shipping a free, reduced version of the SQL Server called Express Edition with their developing IDE Visual Studio and afterwards as stand-alone too. MSSQL is the standard database system used in conjunction with SharePoint Services and .net-development. On the other hand PostgreSQL is the first choice in the spatial data business when working with geological and geospatial data like OpenStreetMap does.

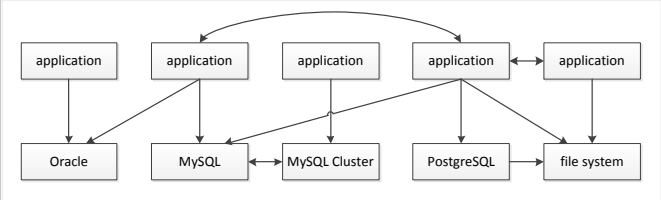
1.1 Differences

Microsoft SQL Server is mainly used for commercial purposes. Many of the main features are not necessary for private applications. This leads to a heavy installation size which in turn brings a long and complex installation process. However, when installed, it offers a very fast database engine which is able to handle a large amount of data and parallel accessing users. But if the Expression edition comes into play, the drawbacks are severe: limitations on database size and only one processor.

Because of the PostgreSQL license – which is similar to the BSD-license – all features are available for everyone. This is probably the most important advantage over MSSQL. PostgreSQL has is strengths in its extensibility and its spatial data handling. Developers are free to include own features in the PostgreSQL source and compile it by themselves.

1.2 Use Cases

Figure 1a: What we don't want (Eisentraut, 2009)



In the situation where both systems are in action, it'd come in handy to link them together. If, for example, a SharePoint environment requires the usage of MSSQL but the other systems need features only available in PostgreSQL, this is very possible scenario. When SharePoint data records are requested by the other systems, they need somehow to be accessible from the outside. Even better, when tables from MSSQL could be used in SQL queries to a PostgreSQL system – or vice versa. There are also very likely scenarios in which spatial data records are stored in PostgreSQL tables while needed on a Microsoft SQL server system – for instance in the mentioned SharePoint environment. Both systems meet these demands. Another unsolved case is accessing non-SQL data like reading from the file system or cooperating with NoSQL-databases like MongoDB.

The solution comes in form of a new standard: SQL/MED. It regulates the Management of External Data in Database Management Systems and could be the much-anticipated solution for the above mentioned scenarios – and even more.

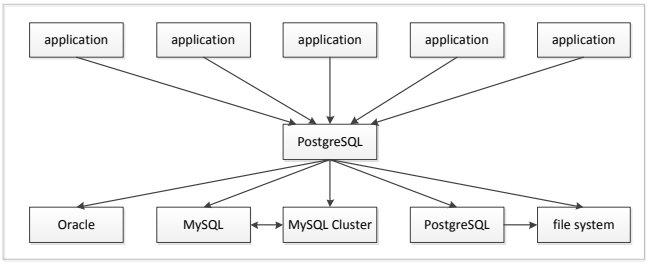


Figure 1b: What we want (Eisentraut, 2009)

2 Interconnectivity between DBMS

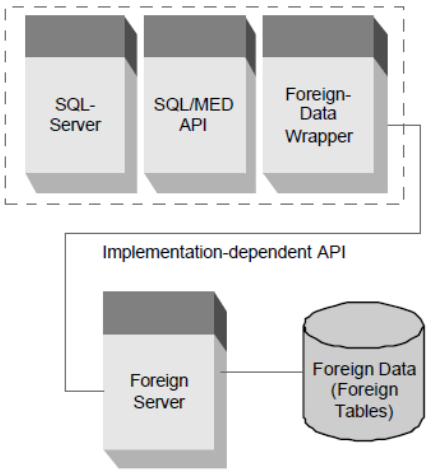
Like mentioned in the introduction, sometimes a link between different database management systems is needed. For the past few years, software producers have included proprietary technologies which enabled connecting to database servers made by their competitors. Examples include Linked Servers (Microsoft), DBLink (Oracle) or custom tools, which link these servers by using ODBC.

2.1 SQL/MED as standard

In order to bring together these technologies and workarounds, a standard named SQL/MED was defined. MED or Management of External Data is an extension to SQL defined by ISO/IEC 9075-9:2003 (Wikipedia, 2011).

- On one hand, it provides so-called *foreign data wrappers*, which access data stored in external data systems. Both SQL-based database systems and NoSQL-databases can be accessed via these data wrappers.
- On the other hand, SQL/MED defines a second technology called *datalinks*, which offer treating external linked files as cell values (Jim Melton, 2002). More information about *datalinks* follows in a later chapter.

Figure 2: MED architecture (Jim Melton, 2002)



Unfortunately, until now, SQL/MED is not very widespread. In fact, only IBM DB2 does fully support the SQL/MED standard while Farrago supports big parts. MySQL seems to implement some syntax elements. Apart from that, no other DBMS implementation of the standard is known. With version 9.1, PostgreSQL makes a big step forward. Many features were integrated. While the extension system is included in the current production release, it still seems unfinished and experimental.

The biggest problem remains the fact that the SQL/MED API seems to be so complex that the available implementations like Farrago (Farrago SQL/MED Support, 2011) and PostgreSQL (PostgreSQL SQL/MED, 2010) use their own API.

2.2 Other technologies

As is so often in software related fields, several producers introduce completely different approaches and technologies. This required more learning for administrators in order to properly use different products. In terms of linking database servers, there mainly are two established technologies: Linked Servers by Microsoft and DBLink by Oracle. The former is able to access all databases, which support the OLD DB programming interface. Thanks to ODBC (which provides OLE DB) mostly all existing databases can be connected.

The Oracle counterpart was mainly built to access remote Oracle databases, but support for products by other vendors and ODBC were also added. It's a very similar approach like foreign data wrappers used in PostgreSQL. For vendor-specific technologies, the usage of Oracle Heterogeneous Services is required. These services implement access to foreign systems. They support the most important database management systems and if not implemented – as mentioned above – ODBC is the way to go.

Even in PostgreSQL, foreign data wrappers are not the first try to include SQL/MED. There has been a project named DBI-Link, which implements parts of the SQL/MED standard (DBI-Link, 2004).

3 Linking PostgreSQL and MSSQL

3.1 Using Linked Servers (MSSQL)

Microsoft's own technology for connecting external database management systems is called *Linked Servers*. This feature was introduced in SQL Server 2005 and enables access to all databases which support OLE DB. Linked Servers support (distributed) queries (SELECT), updates (UPDATE and DELETE) and transactions across multiple database systems.

Abstraction(s)

Like mentioned before, any database server supporting OLE DB can be used by Linked Servers. OLE DB stands for *Object Linking and Embedding Database* and is an API designed by Microsoft. Apart from that, it doesn't have much in common with the OLE (also by Microsoft) technology itself. A set of interfaces implemented using the Component Object Model (COM) builds the OLE DB API. While ODBC only supports SQL-based database systems, OLE DB is located on a higher level which brings support for a wider variety of non-relational databases, such as object databases and spreadsheets (OLE DB, 2007).

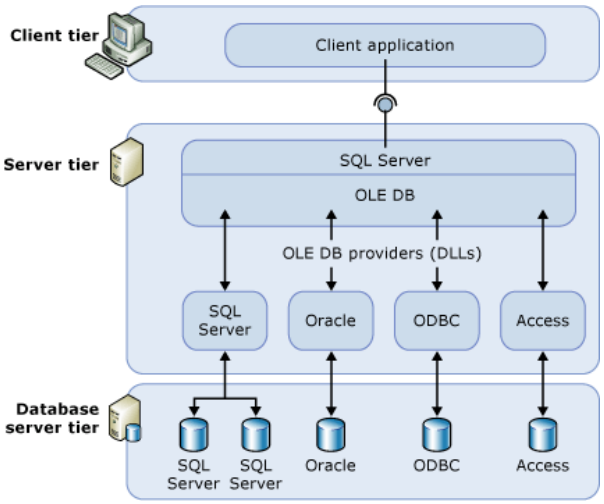


Figure 3: Microsoft Linked Servers (MSDN, 2011)

In the matter of external database systems, OLE DB brings support for Microsoft Access and Oracle. Apart from that, it supports access to any ODBC configured database. This brings a much wider range of supported DBMS, including PostgreSQL. ODBC (Open Database Connectivity) in turn is a standard C interface for accessing database management systems (ODBC, 2011).

Setup

First off, a DSN (Data Source Name) needs to be registered in the Operating System which contains the connection string used to access the PostgreSQL database. Alternatively, it can directly be passed to the Linked Server configuration (italic part).

```
EXEC master.dbo.sp_addlinkedserver
@server = N'myServer', @srvproduct=N'PostgreSQL', @provider=N'MSDASQL',
@provstr=N'Driver=PostgreSQL;uid=user;Server=pghost;database=pgdb;pwd=pw'
```

After the server was added (it's automatically persisted, so it's a onetime thing), queries are written just like normal queries on a local data source. The following examples show a SELECT and a UPDATE query.

```
SELECT * FROM OpenQuery(myServer, 'SELECT * From some_table')

UPDATE OPENQUERY (myServer, 'SELECT name FROM some_table WHERE id = 101')
SET name = 'ADifferentName';
```

3.2 Using Foreign Data Wrappers (PostgreSQL)

Postgre's own implementation of the SQL/MED standard is called FDW which stands for Foreign Data Wrappers. The concept is very similar to ODBC, but allows access to non-SQL database systems. All following commands are defined in the SQL/MED DDL. In this chapter, we'll illustrate the FDW technology using the (still experimental) ODBC_FDW which is available publicly (ZhengYang, 2011).

3.2.1 Install

When using foreign data wrappers, it all starts with a library that handles the communication with the external database system. Ideally, each manufacturer would ship his library amongst his product. In practice, hardly any of the big ones offer such a library – so much to that subject. If such a library is present, the communication flow is pretty straight-forward. It starts with creating the wrapper:

```
CREATE FOREIGN DATA WRAPPER odbc_fdw
LIBRARY 'odbc_fdw.so'
LANGUAGE C;
```

PostgreSQL communicates with `odbc_fdw.so` using the SQL/MED API – in theory. That is a programming-language-neutral, but very complex interface. This is why a PostgreSQL-specific C-interface is used (PostgreSQL SQL/MED, 2010). The library itself accesses the external database using its own ODBC methods. Basically, the library acts as proxy between PostgreSQL and the external database system. In the current version, the FDW has to be compiled and manually installed. Once that was done, the extension has to be loaded:

```
CREATE EXTENSION odbc_fdw;
```

This command looks for a file named "odbc_fdw.control". Information provided in this file includes the path, where the files of the extension are located.

3.2.2 Connect

Creating a virtual "server" is the first step for each usage of a foreign data wrapper. It basically introduces the external database to the local PostgreSQL instance. The command specifies the name of the "server" and the FDW used for access. Beyond that, for ODBC we need the name of the DSN.

```
CREATE SERVER odbc_server
      FOREIGN DATA WRAPPER odbc_fdw
      OPTIONS (dsn 'myDsn');
```

Additionally, authentication information is provided via user mapping.

```
CREATE USER MAPPING FOR postgres
      SERVER odbc_server
      OPTIONS (username 'sa', password 'sa_password');
```

At this point, the external database system is linked. All needed now is a virtual "table" which delivers the rows from the linked database resp. table.

3.2.3 Access

A virtual table gets it data rows from the external system and makes it available for SQL queries sent to the PostgreSQL server. It contains columns like other tables, but it doesn't store the data locally. The *create table* command is composed of different important parts:

```
CREATE FOREIGN TABLE odbc_table (
      db_id integer,
      db_name varchar(255)
)
SERVER odbc_server
OPTIONS (
      database 'exdatabase',
      schema 'dbo',                                                    A
      table 'extable',
      sql_query 'select id, name from `dbo`.`extable`',
      sql_count 'select count(id) from `dbo`.`extable`',               B
      db_id 'id',
      db_name 'name'                                                   C
);
```

- The A part contains the mapping to the database, schema and table on the external system.
- In B, the *sql_query* specifies which fields on the linked table are accessed while the *sql_count* gives information about the numbers of rows returned.
- Lastly, C maps the returned fields to the virtual columns specified in the virtual (local) table. At first the "local" name and afterwards the "remote" name. Looking at the code, it seems that the returned values get casted to strings and then directly casted. In other words: be very careful when mapping the columns and ensure conversion is possible.

The foreign table odbc_table is now accessible via ordinary SQL queries. It appears to the DBMS as a regular table and can be included in commands like JOIN.

3.2.4 Behind the curtain

What happens when a foreign data wrapper is loaded and executed? Let's take a look at the source code. A foreign data wrapper mainly consists of two parts: the handler function and the callback block. PostgreSQL needs to know, how the methods are named which it has to use to access the data in the external system. This information is provided by the handler function. It returns callbacks to all the functions which are used for data access.

```
odbc_fdw_handler(PG_FUNCTION_ARGS)
{
    FdwRoutine *fdwroutine = makeNode(FdwRoutine);

    fdwroutine->PlanForeignScan = odbcPlanForeignScan;
    fdwroutine->ExplainForeignScan = odbcExplainForeignScan;
    fdwroutine->BeginForeignScan = odbcBeginForeignScan;
    fdwroutine->IterateForeignScan = odbcIterateForeignScan;
    fdwroutine->ReScanForeignScan = odbcReScanForeignScan;
    fdwroutine->EndForeignScan = odbcEndForeignScan;

    PG_RETURN_POINTER(fdwroutine);
}
```

When PostgreSQL has the handles to these functions, it uses them to access the external system. These functions can be divided in the following roles:

Table 1: Callback functions

Functions	Role
`Plan`	Estimates the expense needed for the data access.
`Explain`	Possibility to give information about the data source.
`Begin, Iterate, Rescan, End`	Execution of the data access. It starts with *begin*, then *iterates* through the rows and concludes with *end*. A *rescan* would start from the beginning.

In case of the ODBC driver, one of the most interesting parts is the `odbcBeginForeignScan` routine. Therein sits the code which establishes the actual connection to the external database system (via ODBC). For this purpose it uses additional header files named "sql.h" and "sqltext.h" which are the standard ODBC include files (ODBC Programmers Guide, 2002). These are available in ODBC packages like unixODBC (for Linux) or the SQL Server SDK by Microsoft (for Windows).

The source codes shows, that the logic of foreign data wrappers is very simple: they access some data and translate it to a common, standardized format. Basically, the mentioned method creates a connection string and tries to connect to the configured server:

```
appendStringInfo(&conn_str, "DSN=%s;DATABASE=%s;UID=%s;PWD=%s;",
                            svr_dsn, svr_database, username, password);

ret = SQLDriverConnect(..., (SQLCHAR *) conn_str.data, ...);
```

Even though the rest of the code is quit complicated because of the need to standardize the external data rows, the process itself is pretty straight-forward. Connect, give some information about the external data source and deliver the data records – that's the procedure behind all foreign data wrappers.

Status in PostgreSQL

Despite many available implementations of foreign data wrappers in PostgreSQL, only one (*file_fdw* for accessing files as tables) is stable and included in the production release. All the other wrappers were created by different, not necessarily connected developers. The following list gives a quick overview over the most important implementations.

SQL

Oracle	Access to Oracle
MySQL	Access to MySQL
ODBC	Every ODBC supported DB

NoSQL

CouchDB	Access to CouchDB
Redis	Access to Redis

Others

File	Treat files as tables
Twitter	Get data from Twitter.com
LDAP	Query LDAP sources
WWW	Access to RESTful webservices

4 Advanced topics related to SQL/MED

In this chapter we'll go one step further and take a look at the more advanced features of SQL/MED resp. of the implementation in PostgreSQL.

4.1 Foreign Data Wrapper features

Since the first version of the SQL/MED specification, more and more interesting features have been integrated in the new versions. Two of them are presented in the following paragraph.

Complex queries

The old interface of SQL/MED didn't allow an SQL server sending a WHERE clause to a foreign data wrapper. Naturally, this lead to more traffic, because the wrapper sent all rows while the receiving server had to do the filtering. Let's take a look at the following example:

```
SELECT resume FROM emp WHERE name = 'John Doe';
```

Figure 4: Filtering - a delicate matter

Although it's a very simple query, a data wrapper wasn't allowed to handle the filtering. It received the query without the WHERE clause. The current version of the FDW specification does now support more complex queries – like the previous example. This will reduce the amount of data exchanged between the server and the wrapper tremendously. Not to mention, that the wrapper most certainly does know a quicker way to filter his data – the SQL server is only able to iterate all rows and compare the values. This doesn't mean that every wrapper *must* handle complex queries – they can still refuse such queries. In that case, it's again the duty of the SQL server.

Query costs

Most modern SQL servers plan queries by examining different execution plans (Jim Melton, 2002). The time needed of each plan is estimated which leads to executing the one with the shortest execution time. Cause of the importance of this feature, it was integrated in the SQL/MED specification. To illustrate the concept, consider the following example.

```
SELECT city.latitude, city.longitude FROM emp, city
WHERE emp.city_id = city.id;
```

There are several ways for the SQL server to execute this query. If every employee is assigned to only one single city, letting the foreign data wrapper handle this query should be an efficient option (if the wrapper is able to perform joins). If however, for each employee there are 10 cities (we'd need a connection table), the result size would be up to ten times the size of the both affected tables. In that case, much less data would be transferred if the SQL server does the join. These are only two possibilities – in reality, there are many more. To help the SQL server decide about the fastest query, foreign data wrappers can estimate the costs of a query and pass this information to the SQL server.

4.2 Data links

Another very interesting concept is the new DATALINK type. This feature allows referencing files from tables. Thanks to this, big files like images can be externally saved but support all regular database features like access control and integrity mechanisms (Eisentraut, 2009).

Use Case

Certain types of data are primarily stored in files. These types include images and videos. Handling these types by a database management system is very inefficient. The best solution is to store them externally and reference them via a link in a special column. Problem is that these files are isolated and disconnected from the DBMS. Another user on the server could rename or delete these leading to a chaos and a database server which won't ever find his files again.

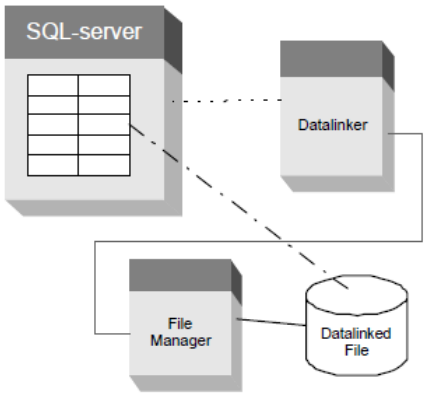


Figure 5: Datalinked file (Jim Melton, 2002)

Concept

The DATALINK type puts the database server out of his misery. It has the ability to create a stronger link to a file and tell the running operating system that the database server is the only "manager" of this specific file. There are a lot of parameters, but the following give a pretty good idea of the concept behind data links:

```
Create Table Rental (
     Video_ID       Integer,
     Clip           Datalink (40)   File Link Control    Integrity All)
```

File Link Control ensures that only existing files can be referenced. Thanks to *Integrity All*, reference files can only be renamed or deleted through SQL. Naturally, data links require a tight coupling of the database server and the OS the server is running on. The implementation is very challenging especially when facing different operating systems and file systems. While data links have not yet been implemented in PostgreSQL, IBM DB2 includes this feature – apparently also in Windows NT – but it's unclear whether more modern operating systems are supported (IBM Research - Almaden, 2001).

4.3 Interesting applications of foreign data wrappers (PostgreSQL)

The Foreign Data Wrapper technology allows innovative add-ons for database systems. Two of them are described in this paragraph. *But please note*: these are experimental implementations and should in no case be used in production.

pgsql_fdw

It may seem obvious to create a wrapper which can access other PostgreSQL instances. However, that functionality was integrated in PostgreSQL quite a long time ago – called *dblink* (not to be confused with *DBI-Link* mentioned earlier). This extension supports querying remote PostgreSQL servers but uses an own API. Apparently, an SQL/MED like syntax was added later – although *dblink* remained to be a standalone extension.

This is why some Postgres developers want to create a new, "true" foreign data wrapper called *pgsql_fdw* (pgsql_fdw, FDW for PostgreSQL server, 2011). This module would be a good example for foreign data wrappers and would also provide the ability to link other PostgreSQL servers via the FDW technology.

www_fdw

Another great example which shows the possibilities of the SQL/MED technology is the *www_fdw* data wrapper. It provides the ability to work with data from web services using SQL. Basically, this wrapper connects to a *RESTful* web service and maps the SQL query to the corresponding HTTP method (www_fdw - Documentation, 2011). It supports selecting data using different criterias (filters, sort and aggregation) and joining the returned rows with other tables. Thanks to RESTful web services, this wrapper works with all web services and doesn't need any special implementations.

In practice, using this wrapper is pretty easy. Let's take a look at an example – we want to search using Google. First off, we need to install the extension and create a *SERVER*.

```
-- Install the extension
CREATE EXTENSION www_fdw;

-- Create the foreign server, a pointer to the web service.
CREATE SERVER google_server FOREIGN DATA WRAPPER www_fdw
OPTIONS (uri 'https://ajax.googleapis.com/search/web?v=1.0');
```

Next step is to create a foreign table which delivers the rows from the external data source. When using *www_fdw*, such a table contains request and response fields (explained in the next step).

```
CREATE FOREIGN TABLE google_table (
  title text,        /* response fields (all three) */
  link text,
  snippet text,
  q text             /* request field */
) SERVER google_server;
```

When executing a query, request fields specify the information sent to the RESTful web service. In contrast, response fields will contain the resulting rows.

```
select * from google_table where q='cat dog' limit 1;

       title       |          snippet          |         link
-------------------+---------------------------------------------------
CatDog - Wikipedia | CatDog is an American...  | http://en.wikipedia...
```

Thanks to the www-wrapper, querying web services from an SQL server is an easy task. No matter which service should provide the information, we'll never need a new implementation.

5 Conclusion

While the SQL/MED DDL provides a fast and easy way to access external data, the implementations behind them need much work in order to become a more widespread standard.

5.1 Handling

Creating a foreign data wrapper is far from an easy task. But when a wrapper is available, the SQL/MED DDL is very easy to use. Only few steps are needed: establish a link (CREATE SERVER), define the query (CREATE FOREIGN TABLE) and select rows from that "table". It's an intuitive process and a great concept. In an environment where different database management systems are present, we are in need of a comprehensive standard which offers a flexible system to link these different database servers.

This was the idea behind SQL/MED. After witness various vendor-specific implementations - which had their own advantages and disadvantages, but sure as death were not compatible – come, defining a public standard was definitely the way to go. Although SQL/MED has yet to establish itself in the DBMS world, it should only be a matter of time.

Additionally, data links – the other concept defined in SQL/MED – are absolutely one of the best innovations in the SQL standard. They fill a painful gap in the daily usage of SQL. Big files saved in SQL tables always lead to slow queries. Storing the files externally was the only solution, but it was problematic. Missing control over the files led to problems, which included deleted, renamed or externally changed files. Data links finally fix this issue.

5.2 Status

No matter how well the SQL/MED DDL was defined, the SQL/MED API (which accesses the library files) is far too complex and non-trivial to implement. Sadly, this fact led to custom, vendor-specific implementations of foreign data wrapper libraries. For example, PostgreSQL uses the previously showed interface to access its libraries. On the other hand, Farrago resorts on Java. Actually, different languages per se aren't the problem. It's the huge interface that needs too much implementation-work. Because each vendor would like to have as much developer-support for his product, they each created newly designed interfaces, which can easily be implemented.

5.3 Outlook

Certainly, while this trend remains, the chances for SQL/MED to establish itself on the market aren't exactly perfect. The main problem is the lacking exchangeability of the developed foreign data wrappers. Even though more and more foreign data wrapper libraries are popping up – most notably for PostgreSQL, but also for Farrago – the "standard" SQL/MED can't benefit. Each wrapper needs a different implementation for all the database management systems that support SQL/MED. The best way to handle this issue would be to create a new API that can easily be implemented by the developers. Until that will happen, SQL/MED – a great concept – will have to wait for its breakthrough.

6 References

IBM Research - Almaden. (2001, February 23). Retrieved December 20, 2011, from IBM: http://www.almaden.ibm.com/projects/datalinksfaqs.shtml#q1

ODBC Programmers Guide. (2002). Retrieved December 02, 2011, from DBMaker: http://www.dbmaker.com.tw/reference/manuals/odbc/odbc_chap_02.html

DBI-Link. (2004). Retrieved November 27, 2011, from Project Overview: http://dbi-link.projects.postgresql.org/

OLE DB. (2007, April 07). Retrieved November 22, 2011, from Wikipedia: http://en.wikipedia.org/wiki/OLE_DB

PostgreSQL SQL/MED. (2010). Retrieved November 11, 2011, from PostgreSQL Wiki: http://wiki.postgresql.org/wiki/SQL/MED

Farrago SQL/MED Support. (2011). Retrieved December 10, 2011, from Farrago: http://farrago.sourceforge.net/design/sqlmed.html

MSDN. (2011). Retrieved November 19, 2011, from Linking Servers: http://msdn.microsoft.com/en-us/library/ms188279.aspx

ODBC. (2011, March 02). Retrieved November 23, 2011, from Wikipedia: http://en.wikipedia.org/wiki/ODBC

pgsql_fdw, FDW for PostgreSQL server. (2011, October 25). Retrieved December 12, 2011, from pgsql-hackers: http://archives.postgresql.org/pgsql-hackers/2011-10/msg01329.php

www_fdw - Documentation. (2011). Retrieved December 12, 2011, from github: https://github.com/cyga/www_fdw/wiki/Documentation

ZhengYang. (2011). Retrieved October 17, 2011, from Github: https://github.com/ZhengYang/odbc_fdw

Eisentraut, P. (2009). *PGCon.* Retrieved October 22, 2011, from SQL/MED: Doping for PostgreSQL: http://www.pgcon.org/2009/schedule/attachments/133_pgcon2009-sqlmed.pdf

Jim Melton, J.-E. M. (2002, September). SQL/MED — A Status Report. *SIGMOD Record*, pp. 81-89.

Wikipedia. (2011, January 3). *SQL/MED*. Retrieved November 24, 2011, from Wikipedia: http://en.wikipedia.org/wiki/SQL/MED