



Making Data Available on the Web

SimbaEngine ODBC SDK

By Simba Technologies Inc.

Introduction

Many companies use web-based services to automate business processes like sales, track items like packages, or sell things. These web-based services usually include a web front-end that allows the user to enter information, perform functions, create reports, and provide security and authentication. Behind the web site is usually a database that contains the information for all the web service users. This database may be a relational database, but it might be non-SQL.

For normal functions, the web site provides everything needed by the users. They may even be able to customize business processes or the database schema. However, for many of these sites there is a limit to what the users can do. The sites are designed to serve the majority, but sometimes a minority wants more functionality in an area like reporting. The canned reports available on the site will fulfill most of the common reporting needs, but there are always a few complicated reports that users struggle to create. Sometimes they can't be created at all.

The frustrating thing about limits on reporting is that there is an obvious missing piece – a secure ODBC or JDBC connection using SQL – that would solve the problem if the web site supported it. Perhaps the users want to join three tables, or pull out all of the unique values in a field, or group and sum records. ODBC and SQL can do these things trivially, but many web services sites can't.

What is the solution?

Data is everywhere on the Internet. It is public, private, visible, hidden and for sale. Often it is right in front of us and we don't notice it. What it mostly lacks is availability for analysis and reporting by common and familiar applications, and that can be a problem. You might have this problem if you have data on your web site and your customers can't access it with Microsoft Excel. Your data could be trapped.

When we think of "data", we often think of highly structured

data in relational databases. The data on the Internet is sometimes in this form, but more often it isn't. In any case, relational data is not usually exposed directly to the Internet for a variety of good reasons. Much of the data on the Internet is hiding in plain sight behind the applications that provide access to it. Some of the most-used databases on the Internet are Google and Amazon.com, but other familiar ones are eBay, Salesforce and FedEx. There are thousands of similar sites of varying sizes, but they all share the attribute of a database behind a web application.

The Situation

The issue of trapped data has several dimensions. Looking at these dimensions allows us to understand the situation more clearly so we can see what can be done. The main dimensions are:

- How the data is accessed (web services, proprietary protocol),
- How the data is searched (SQL, non-SQL language, key-value, other), and
- What the data is for (support for a web application, proprietary data for sale, data storage for rent).

Data Access: A network protocol is required to access data remotely. Web services use HTTP, which makes getting to where the data is very easy, but HTTP is not efficient for moving large amounts of data. Or, users can use an efficient proprietary protocol to access the data. Traditional relational databases use these and they are very fast, but the whole system of data access can be rigid, closed and not appropriate for a multi-tenanted database.

Search Method: How users find the information they want in the database is the next issue. Relational databases use SQL. There are lots of less-well-known alternatives, including XQuery, LINQ and SPARQL. These query methods allow users



to specify complex search criteria and retrieve the records they want. Another technique that is growing in popularity is key/value indexing of the data. This allows very fast data searches at the expense of expressiveness, but sometimes that is a worthwhile tradeoff.

Purpose: The purpose of the web database is perhaps its most interesting characteristic. We can break this down into at least three big groups:

1. Support for a web application,
2. Proprietary data for sale, and
3. Data storage for rent.

Support for a web application is perhaps the most common use of Internet databases. In this category are sites like Amazon.com, eBay, Salesforce and FedEx. These sites use a combination of web applications and a database to provide service to their customers. The web applications provide access to the data, but restrict how it can be used. It's hard to get to the data without the application.

There are many vendors of data on the Internet, from giant Axiom to individual Multiple Listing Services (MLS). Sometimes this data is available as downloaded files, and sometimes it's available through an application. However, a downloaded a file isn't up to date, and web applications provide the data only in the ways its developers imagined.

One of the most discussed Internet topics today is cloud computing, which includes data storage in the cloud. The very biggest names on the Internet are involved in providing services for storing data in the cloud - Google, Amazon and Microsoft. The choices are wide and there are always tradeoffs.

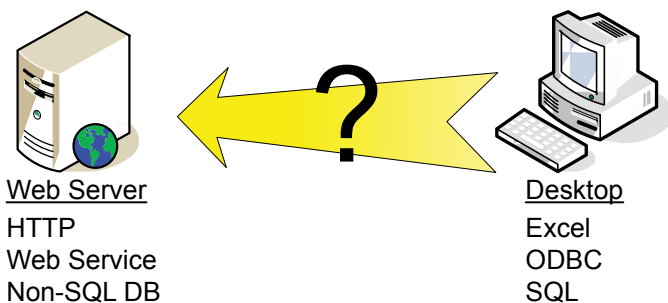
The Problem

All this leads us to the problem with much of the data on the Internet: If a user is interested in more than a few dozen records, they may find that the available web applications don't provide all the analyses they want. Most of their work is still done using the available web applications, but they

want to complete the suite of functionality by adding custom analysis. But while they may know exactly how they want to analyze the data with their favorite desktop application, they can't get the data to the application.

There are two important reasons that data consumers might want to use desktop tools to analyze Internet data. First, users are often very familiar with common desktop reporting and analysis applications. Most people have Microsoft Excel on their computer at work, and a significant number will have SAP BusinessObjects Crystal Reports. These are capable tools that users invest time and effort to learn, and they feel comfortable using them. Further, when someone is familiar with a reporting or analysis tool, they will also be more trusting of the results it gives, especially if the results are startling. Since results that are truly startling are the ones users need to know about, this is an important consideration.

Second, applications like Microsoft Excel and MicroStrategy are powerful and sophisticated analysis tools that provide a wide range of capabilities. Remember that Microsoft Excel PivotTables is the world's most popular Business Intelligence (BI) tool. It is likely that there are operations and analyses familiar to end users from these tools that are simply not available in canned web applications. Developing and verifying sophisticated analysis tools, especially general purpose ones, is an expensive business. It is unlikely that the developers of custom analysis web applications will duplicate all of the functionality of Excel. Instead, they will provide some relevant analyses and leave it at that. When users want more, they have to look elsewhere.

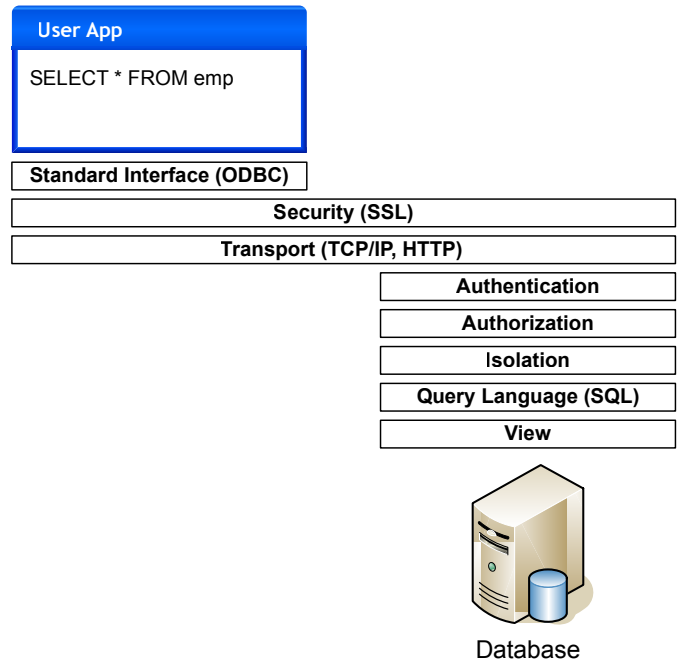




Common desktop applications access data in a variety of ways. The most common are the ODBC, JDBC, OLE DB and ADO.NET standards. If users can download their data as .csv (comma-separated-values) files, they can import it into most reporting and BI tools. But this process adds complexity, and the data is no longer live. What users really want is standards-based access to live data so they can connect their favorite application directly to the database and get to work. The question becomes, "how do you bridge the gap between Internet data and desktop reporting and analysis tools?"

The Solution

Simply put, the solution is to provide a standards-based (i.e.: ODBC, JDBC, OLE DB or ADO.NET) interface to your Internet data to complement the existing database, processes and web applications already available. Then, end users can connect and query their data with desktop applications when it is appropriate. But this statement hides a multitude of problems. Perhaps a way to make this solution easier to understand is to break it down into sub-problems. This reduces the size of each problem as we think about it, and we may discover that we already have part of the solution and that we only need a few missing pieces to deliver significant value to end users.



Here is one way to break down the problem of connecting desktop applications to Internet data. In any solution, all of these functions have to work together quickly and reliably, but they can be supplied by different vendors and systems depending on how the database is deployed.



Application: The desktop application has to have a standard interface for querying data: ODBC, JDBC, OLE DB or ADO.NET.

Standard Interface: If we assume the application has a standard interface, then we have to provide a compatible standard data driver. This means providing an ODBC, JDBC, OLE DB or ADO.NET data driver for your Internet data. A standard interface means there is effectively no limit to the range of applications that can be used.

Security: In this context, we are talking about the security of the information moving across the Internet – both identity and command information from the application and the retrieved data from the database. Secure Sockets Layer (SSL) can provide the required security.

Transport: Of course, to move data from an Internet-based database to a workstation requires some form of transport. This can be HTTP, if the data is available via a web service, or a proprietary protocol, if it works well across the Internet and through firewalls.

Authentication: When a connection is made from the workstation to the central server, the user's identity has to be verified to make sure they are allowed to access the data. For freely available data, this may not be necessary, but for proprietary or sensitive data this might involve some very sophisticated security measures.

Authorization: Once the database system has determined who the user is that it has to validate, they are allowed to see the data they are requesting. This can be driven by a set of simple role-based rules, or it can be very complicated with rules that look at the characteristics of the returned data. For multi-tenanted use, this is typically more than commercial database systems provide.

Isolation: "Will my data be protected from access by others using the system?" This is a slightly different take on Authorization and is particularly acute in multi-tenanted systems. If everyone's data is mixed in together, it requires very strong isolation to maintain the separation and to maintain the trust of your users.

Query Language: A database requires some form of query specification so that users don't get *all* of the data every time. SQL is the most used query specification (language), and it's commonly used by popular reporting tools. There are other languages (e.g.: XQuery, SPARQL) and methods (e.g.: LINQ) for specifying what data is wanted from a database.

View: Traditional SQL databases implement the concept of a view. This is a notional window into the database that shows only a subset of what is there. Views can be used to restrict access to data, but they are often role-based, which makes them clunky for a multi-tenanted database. Views are fine for a single organization that has Administrator, HR, Manager, Analyst, Data Entry and Clerk roles, but they are less suitable when you have 500 customers who each have those roles.

Data: Finally, there is the data itself. It has to be organized and classified in a way that will support the requirements of the issues above.

Discussion

All of these issues have to be addressed. Making an ODBC driver available so customers can use Excel with a simple web services database is simpler than creating direct access to a multi-tenanted relational database. But in either case, the task is smaller and easier if you start with robust, customizable and capable components that deliver large pieces of technology you would otherwise have to build

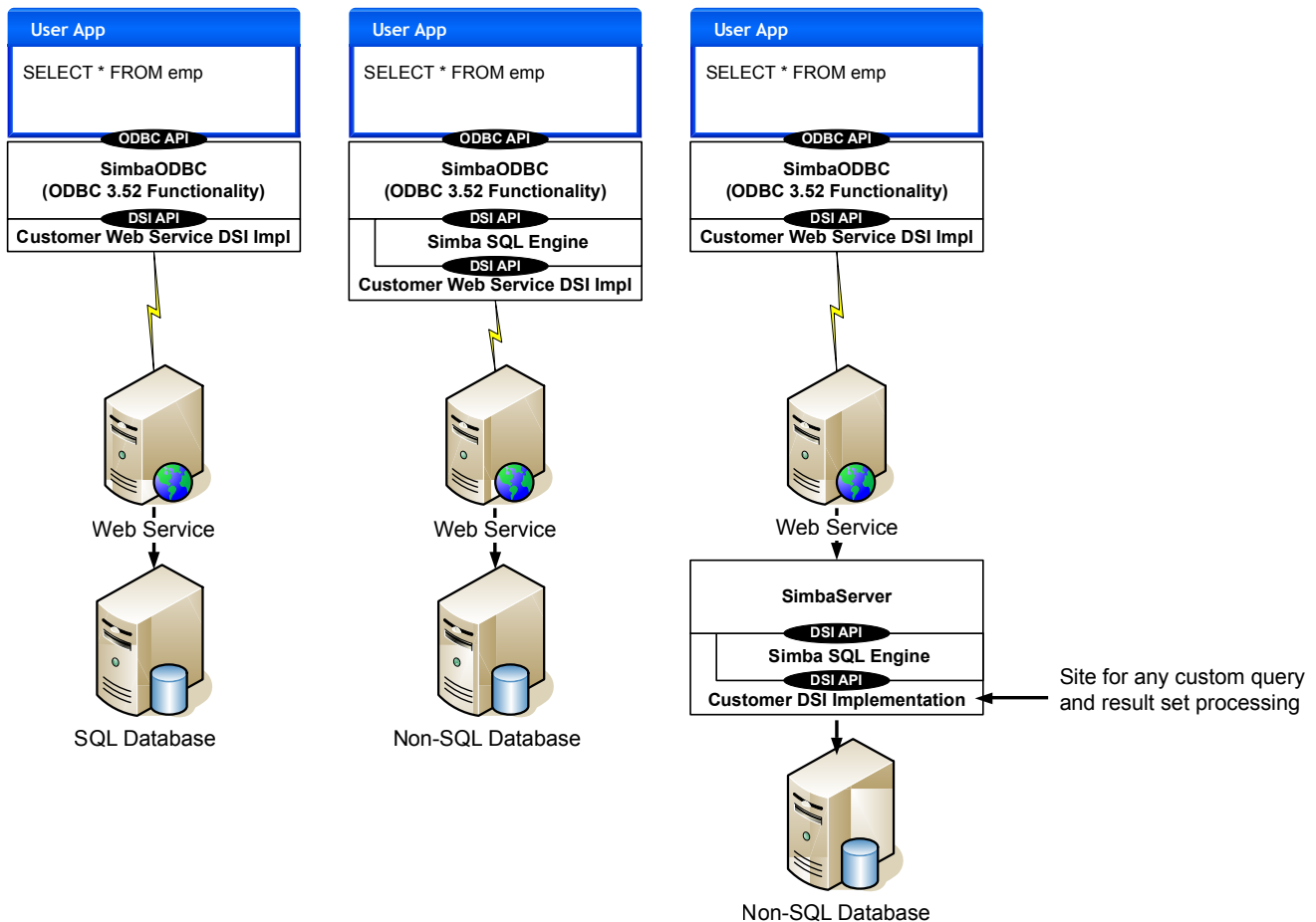
yourself. SimbaEngine SDK contains the technology in a conveniently componentized form to help you efficiently create and integrate solutions for these issues. Here are some suggestions:



Web Services

If you already have a web services interface to your data, and many Internet databases do, then the primary task is to provide a standard interface on the user's workstation that can communicate through the web service to the database. SimbaODBC is a complete ODBC interface component you can use to easily create an ODBC driver to a SQL-enabled web service database. On the workstation side, you will write

a little bit of code that is called by SimbaODBC and makes the connection to the web service. If the web service supports SSL, then you can easily add SSL to your code to secure the communication. After a connection is established, SQL queries can be sent to the database and results retrieved as XML documents (left-hand example below.)



Accessing a SQL database via a web services interface.

Accessing a Non- SQL database via a web services interface. The SQL engine is on the client.

Accessing a Non- SQL database via a web services interface. The SQL engine is on the server.



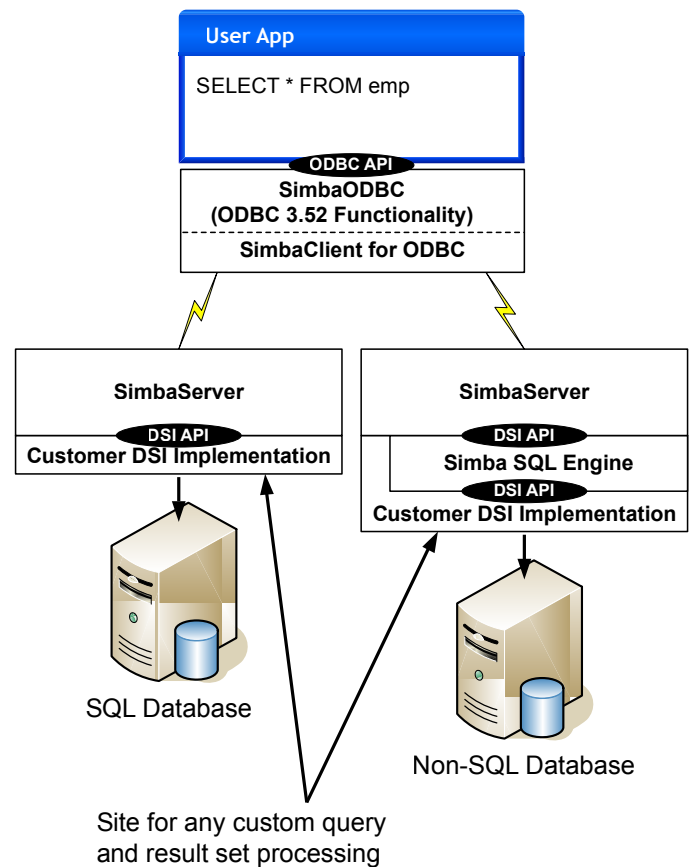
If your web services database does not support SQL, you will have to add it with your solution. SQL processing can either be added to the ODBC client or integrated into the server. Simba's SQL Engine is a complete SQL parsing, preparation and execution engine that works seamlessly with SimbaODBC to create an ODBC driver containing the required SQL processing (middle example above.) In this architecture, you need to reduce the amount of data moved from the web service to the driver on the workstation to keep the performance acceptable. Luckily, most databases, even if they don't support SQL, do support some form of filtering at the table level. Simba SQL Engine implements Collaborative Query Execution that dynamically splits the query processing between Simba SQL Engine and the query processing supported by the database. This allows Simba SQL Engine to move as much query processing as possible to the web services database, reducing network traffic while maintaining SQL compliance.

Simba SQL Engine can also be integrated into the server side of a web services database (right-hand example above.) This can be done by either building a SimbaServer/Simba SQL Engine solution through which the web services server queries the database, or by integrating the Simba SQL Engine directly into the existing web services infrastructure. Simba SQL Engine is componentized so it can be embedded into other processes and deliver the same robust, complete SQL processing.

Non-Web Services

If the database you want to expose does not have a web service interface and you do not want to provide one, there are other options available. Simba Client/Server is a self-contained set of components that provide high-performance, secure, remote data access via standard ODBC and JDBC interfaces. OLE DB and ADO.NET can be added as well. SimbaServer can handle thousands of simultaneous client connections while maintaining efficiency, performance and reliability. The SimbaClients for ODBC and JDBC can be installed easily on your users' workstations to immediately provide standards-based connectivity to SimbaServer. The proprietary protocol is designed for high-performance database access across the Internet and has SSL security built in.

SimbaServer works seamlessly with Simba SQL Engine, if the database is not SQL-capable. This is similar to the scenario above where Simba SQL Engine is used to provide SQL processing on the server side for a web services system. The only difference in this case is that your users' workstations are connecting directly to SimbaServer instead of through a web service. A small amount of custom code connects Simba SQL Engine to the database, but the rest of the technology is in SimbaEngine SDK.





For a SQL-capable database, SimbaServer can connect directly without the intervention of Simba SQL Engine. Once again, a small amount of code connects SimbaServer to the database, but the rest of the technology is available in SimbaEngine SDK.

When SimbaServer is employed to access the database, the code you write to connect them is the perfect location to implement custom authentication, authorization and isolation. At this point in the system, all user credentials are visible as are all queries and result set data. Credentials can be examined, queries scrutinized and modified, and result set data analyzed. It is easy to imagine a complicated set of data access and availability rules custom coded into SimbaServer and driven by tabular data that can be updated daily as end users come and go and change their status. Simba Technologies has the experience and the knowledge to specify and create such systems for you, or to help you develop them yourself.

About Simba Technologies Inc.

Simba Technologies Inc. is the recognized world leader in standards-based data access products and solutions. We work with the world's leading software companies to deliver first class data connectivity solutions.

Simba is a pioneer in ODBC, MDX, OLE DB for OLAP (ODBO) and XML for Analysis (XMLA). Since 1991, we've developed advanced data access solutions for thousands of end users. Today, more than half of all MDX providers have been built with Simba technology, and through a partnership with Microsoft, Simba's SQL technology has been installed on more than 30 million desktops worldwide.

Our firm commitment to delivering the highest customer value through innovative solutions and expert support has gained us a reputation as the industry leader for data connectivity solutions.

©2010 Simba Technologies Inc. All Rights Reserved.
Printed in Canada.

Simba Technologies Incorporated

938 West 8th Avenue
Vancouver, BC Canada
V5Z 1E5

Tel. +1.604.633.0008
Fax. +1.604.633.0004
Email. solutions (at) simba.com
www.simba.com

Simba and the Simba logo are trademarks of Simba Technologies Inc. All other trademarks or service marks are the property of their respective owners.