Data Storage Architecture with SQL Server 2005 Compact Edition

Source	http://msdn2.microsoft.com/en-us/library/bb380177.aspx

Introduction

Selecting the right data storage architecture for your applications can be a daunting task. The number of choices for a data storage technology is large and growing every day. The data storage technology you select depends on a number of factors. You must analyze tradeoffs between platform requirements, size, performance, ease of deployment, ease of data access, and data store capabilities.

For server applications servicing large numbers of users, the choice is clear: use SQL Server 2005. Which server-based edition of SQL Server 2005 you use depends on the scale and focus of your application, but the feature list makes it fairly easy to decide which version you need. Furthermore, changing versions is a simple licensing decision and usually does not require an architecture change.

For client-side applications or small-scale server applications, selecting a data storage technology is a little more challenging. For client applications, it does not make sense to put a full scale instance of SQL Server 2005 on each client computer because of cost, complexity, platform requirements, and a number of other factors. For small-scale server applications, the extra horsepower of SQL Server 2005 may not be needed, and the licensing cost may be prohibitive for small projects. For mobile device applications, a full version of SQL Server is not supported by the platform.

This white paper discusses data storage architecture challenges, scenarios, and solutions using the new SQL Server 2005 Compact Edition (SSCE). It covers the similarities and differences between SSCE, other SQL Server 2005 editions, and other relational database technologies, including the EDB embedded database engine on mobile devices.

Data Storage Challenges

For either client application or small-scale server applications, you need to address a number of data storage challenges:

- **Data storage location.** If you are building a distributed client application and can afford to store your data on a back-end server and retrieve the data over the network, you can just use SQL Server 2005 on the server. If you are building a mobile device or client application, you will likely need a local data store on the client to cache data when you are offline. You may also need to cache on the client to avoid retrieving large datasets, such as a product catalog, repeatedly over the network. For client applications, the data may only be needed locally, in which case storing on a back-end server does not make sense.
- **Ease of data access.** Productivity is a very important part of getting applications to market on time and in budget. Your data storage technology should make it very easy to read and write data from your storage location.
- **Ease of querying.** A robust data storage technology enables you to easily search for and select individual records or collections of records, and do so quickly.
- **Ability to synchronize data stores.** For mobile client applications, the offline data stored locally must be synchronized with back-end data stores. Writing synchronization mechanisms from scratch is error prone and time consuming. The data storage technology should support synchronizing multiple data stores.
- **Security.** Particularly in the case of mobile devices or portable client computers, security is very important in terms of protecting the data when it is stored so that, if the computer is stolen, the data is inaccessible to an unauthorized user. Also, when you synchronize the data, there must be protection for the data in transit.
- **Data integrity.** When you read and write data from your store, you need to be sure the data is stored consistently and that no data corruption occurs. Transactional data stores provide mechanisms for ensuring this, and should be favored over non-transactional data stores.
- **Ease of deployment.** For client applications, a lightweight footprint and easy installation process is critical for supportability and maintainability. The configuration required in the client application should also be minimized to tie the application to the data store.

Data Storage Overview

A range of options are available for data storage. Not long ago, many applications serialized data in proprietary formats to flat files on disk. XML provides a more structured and easy-to-consume way of storing arbitrary data in files, but does not address many of the challenges described in the previous section. Relational database technologies are really the only widely available data storage technology to address the full range of challenges needed for robust data storage.

Even once you have narrowed your focus to database technologies, there are a number of options available. You need to use other criteria to select the appropriate technology, and select the right database technology based on the way each option addresses the challenges discussed in the previous section. In addition, you need to consider whether the database will run as a single-user client database or a multi-user server database.

In client applications, you should narrow your focus to one of two options, depending on whether it is a desktop application or a mobile device application. For desktop applications (running on a desktop workstation, laptop, or Tablet PC), focus on SQL Server 2005 Compact Edition (SSCE) or SQL Server 2005 Express Edition (SSE). For mobile devices running Microsoft Windows CE or Mobile operating systems, you can choose between SSCE or the EDB embedded database engine.

For a number of reasons discussed in more detail later in this white paper, SSCE provides a great and easy-to-use solution for most client business applications. SSE is only needed for specialized client-side applications, but makes a great choice for small-scale server applications that support moderate user loading, and that need a more robust and scalable architecture. EDB may be a good choice if you need native support on the device and are concerned about the disk and memory impact of installing SSCE on the device, but you will have to weigh this against the greater productivity and features of SSCE.

For small-scale server-side applications and caching needs, either SSCE or SSE may be appropriate. For a Web application that has any potential to grow and scale in the future, SSE is a better option because it supports a wider range of capabilities as compared to SSCE.

Data Storage Applications

Applications come in many shapes and sizes. As mentioned earlier, large-scale server applications really need the SQL Server 2005 Workgroup, Standard, or Enterprise editions and are not the focus of this white paper. For client applications and small-scale server applications, you can categorize applications into a number of application types. These types include:

- **Field force applications.** These are client applications used by mobile workers in the field. These applications typically run on portable computers or Tablet PCs, but may also run on mobile devices. They provide interactive data presentation and interaction for working with distributed customers or equipment.

- **Personal information management (PIM) applications.** These are client applications used to store and access collections of information items, such as contacts, schedules, tasks, notes, and so on. They can run on desktops, portable computers, or mobile client devices, but more and more these types of applications are being developed for low-footprint installation on mobile devices.
- **Small-scale Web applications.** These are simple Web presence client applications for presenting dynamic information over the Web to a small audience of users. Examples might be small companies, community and nonprofit organizations, or clubs. These require that a Web server be exposed, but could potentially be run from a desktop operating system such as Windows Vista or Windows XP Professional.
- **Caching applications.** Whether or not a client application is designed to work offline, it usually does not make sense to make a full roundtrip to the database every time you need to obtain lookup information such as city, state, postal code, product catalog listings, and so on. Implementing a client data cache can provide significant performance improvements for presenting lists or searching in lookup tables that change infrequently. Additionally, some applications may be designed with a distributed architecture involving smart client, mobile device, or Web client applications that communicate with a centralized back-end application server. The application server may need to cache data locally to avoid roundtrips to the database server. The need to cache data on the local machine does not warrant the purchase or overhead of installing a full instance of SQL Server locally on the application server.

Data Storage Options

To make good architecture decisions, approach the solution objectively with a good understanding of the available options. For the data storage needs of your architecture, there are a number of options to examine. Again, this paper focuses on client-side and small-scale server data storage. For these parts of your architecture, your primary options are file storage, EDB, SSCE, and SSE. To make the right selection, you must examine the capabilities and limitations of each choice, and weigh them against your requirements.

Flat or XML Files

Using flat files or XML files may seem attractive on the surface from a simplicity standpoint: most developers already know how to use Microsoft .NET or XML serialization to read and write data from files, or can use the capabilities of a dataset to read and write it as XML from a file. However, files have problems in terms of consistency and reliability. If an application is interrupted or shuts down without properly completing I/O operations and closing the file, the contained data can be corrupt and the file lock may prevent the application from behaving properly. These problems can be solved through disciplined coding and error handling, but achieving that removes the simplicity that initially made flat files attractive.

There is also no inherent ability to query files, and you generally must read all or most of the file into memory data structures to make the data usable. As a result, flat files and XML files are not appropriate for data that will be queried, read, and written throughout the execution of the application. For simple data such as configuration settings and preferences, XML configuration files are appropriate. It may also be preferable to store large documents and files as flat files with metadata in the database, depending on the access patterns. But most random access read/write data for your applications should be stored and accessed through a database engine.

EDB

The EDB embedded database engine is a native part of the Windows CE 5.x, 6.x, and Mobile 6.x operating systems. EDB is a simple relational data engine that allows you to store tabular data with a limited type system and limited query capabilities. EDB is an attractive option when you want a capability that is native on a device or that can be installed with a minimum footprint and overhead on the device. EDB can be faster than SSCE for certain kinds of simple data retrieval, but may also be slower if complex query operations are needed. One of the biggest downsides to using EDB is that you only have access to it by using a low-level API in Microsoft Visual C++. If you are writing your application by using the .NET Compact Framework to realize the productivity benefits associated with managed code development, you must write a separate data access component that works with the EDB database using C++, and then you need to interoperate with that component from your managed code. The complexity this introduces likely outweighs the benefits of using EDB. Only if you are already writing your application with embedded C++ and are focused on minimizing your footprint and maximizing performance of the application will you want to consider EDB.

SQL Server 2005 Compact Edition (SSCE)

SSCE is a lightweight (< 2 MB), free relational database engine that can be installed on any current Windows operating system. Because SSCE is the primary focus of this white paper, the full feature set is described in a later section. At a high level, SSCE supports tables, relations, constraints, complex query processing, transactions, replication, and data security. To program against SSCE, you use an ADO.NET managed provider with data access coding patterns similar to what you use for other managed providers, such as the SQL Server SQLClient managed provider. You can also access SSCE from unmanaged clients by using OLE DB. SSCE runs in-process as a set of libraries referenced by the using application, and is easy to deploy either with your application libraries or as a separate MSI install. SSCE can be easily deployed with ClickOnce applications or Xcopy-deployed onto mobile devices. SSCE will also be installed natively on Windows Mobile 6.0 or later.

The SSCE type system is a subset of the SQL Server 2005 type system, and does not support all features that a full SQL Server instance supports. Commonly used features from SQL Server for server applications that are not present in SSCE include stored procedures, triggers, views, functions, user defined data types, and the ability to participate in SQL Server Service Broker messaging.

SQL Server 2005 Express Edition (SSE)

SSE is a fairly small footprint (< 55 MB), free database engine service that can be installed on any current desktop or server Windows operating system. Because SSE runs as a Windows Service, it requires a Windows Installer (MSI) installation on the target machine. SSE can be deployed through the ClickOnce Bootstrapper to allow it to be used by ClickOnce-deployed applications. SSE supports user instance isolation, which can ease ClickOnce deployment by ensuring that one user's data is automatically protected from other users' data.

SSE supports most of the capabilities of a full instance of SQL Server, including tables, views, stored procedures, triggers, functions, and SQL CLR. You access data in a SSE instance from managed code the same way that you do from a full SQL Server instance, using the SQLClient managed provider. You can also access it from unmanaged applications by using an OLE DB provider.

The limitations of SSE compared to a full instance of SQL Server are fairly easy to understand. SSE uses only one processor on the computer even if more are present; it uses only 1 GB of memory; and it only allows databases to grow to 4 GB in size. Additionally, SSE can be a Subscriber but not a Publisher for all types of replication, and can send and receive SQL Server Service Broker messages as long as the messages are passed through a full instance of SQL Server (that is, no peer-to-peer messaging between SSE instances without a full instance in the chain).

SQL Server 2005 Compact Edition Overview

At a conceptual level, SSCE can be thought of as a significantly scaled-down version of the SQL Server 2005 database engine. However, it is a separate database engine designed to maximize the key features needed for simple, secure, and transactional relational data storage while minimizing the disk, memory, and installation requirements for the hosting application.

SSCE History

The heritage of SSCE traces back to SQL Server CE 1.0, which was released in 2001. This was the first Microsoft-released relational data engine for mobile device operating systems, and was based on SQL Server 2000 database capabilities. Versions 1.1 and 2.0 followed to improve the user experience, with 2.0 providing integration with .NET Compact Framework applications. Along with .NET 2.0 and SQL Server 2005, SQL Server 2005 Mobile Edition was released as the next-generation mobile database engine. SQL Server 2005 Mobile Edition offered a lot of new features, including better reliability and performance, better synchronization options, and better integration with both SQL Server 2005 and Microsoft Visual Studio 2005.

When SSCE was first announced at TechEd 2006, the new name for this version was announced as SQL Server 2005 Everywhere Edition. This name only existed for a short while between the first announcement and when Microsoft refined the plans and release features for SSCE and announced those and the new name at TechEd Europe 2006 in November 2006. As a result, you may find some materials that will linger for a while on the Web with the name SQL Server 2005 Everywhere Edition, or SQL Everywhere, for short—treat those as synonymous with SSCE. It is important to note that SSCE is a completely different and significantly more capable data engine than the earlier SQL Server CE editions were, so those need to be carefully distinguished.

SSCE Core Capabilities

The core capability of SSCE is to allow secure transactional relational data access and storage. You can execute SQL queries that include Data Definition Language (DDL) and Data Manipulation Language (DML) queries through the SSCE engine. With SSCE, you create a database instance as a single .sdf file. Within that database you can define tables with primary keys and constraints. SSCE supports full referential integrity through foreign key constraints, and cascading deletes and updates.

Additionally, SSCE supports the following features:

- **Multiple concurrent connections for multi-threaded data access.**
- **Password protection and 128-bit encryption of the .sdf SSCE data file.**
- **A wide range of column data types.**
- **Scrollable, updateable cursors for fast and easy connected-data access.**
- **Databases that can grow up to 4 GB in size.**
- **Synchronization with SQL Server through merge replication and Remote Data Access (RDA).**

SSCE New Capabilities

All of the core features of SSCE were actually part of SQL Server 2005 Mobile Edition, which was released with SQL Server 2005 and .NET 2.0 in October 2005. SSCE adds a number of peripheral features that distinguish it from the Mobile Edition:

- **SSCE can now be run on any supported Windows operating system, including mobile devices, Tablet PCs, portable computers, desktops, and servers.**
- **SSCE can be deployed by using ClickOnce.**

Data Synchronization with SSCE

When SSCE is used as a local data cache for a client or middle-tier application in a distributed application architecture, you usually need to be able to synchronize the SSCE database with a back-end database server. You may need to populate the SSCE database tables initially from a back-end database, and you may also need to be able to push updated or new records from the client to the server database.

With SSCE, you have a number of options for synchronization. There are two built-in synchronization options: merge replication and RDA. Both of these capabilities allow you to synchronize data in both directions: from an SSCE database to a SQL Server 2005,. or SQL Server 2000 database over HTTP. Besides these capabilities, you could implement a custom synchronization solution by making calls to your own exposed Web service endpoints, allowing you to pass the data through custom business logic layers in both directions on the server side. Also, coming in the near future with the next release of Visual Studio (code named Orcas), there will be a new synchronization subsystem called the Occasionally Connected Systems (OCS) synchronization framework.

Merge replication is the most powerful built-in synchronization option because it allows the autonomous and independent updating of records on both the client (SSCE) and server (SQL Server) databases at the same time. SSCE participates in merge replication as a subscriber, and can subscribe to merge replication publications exposed from either SQL Server 2000 or SQL Server 2005. Merge replication handles many clients easily, because it also supports conflict detection and resolution on the server side. Merge replication is a little more complicated to set up, and requires specific database design features and configuration on the server side.

RDA is the easiest built-in synchronization option to use. With RDA, there are no specific requirements on the server-side database. To synchronize data with RDA, you pull an entire result set into the SSCE database from the server side to initialize a table with the current data from the server. You can then push changes (inserts, updates, and deletions) back to the server when you choose to synchronize. To obtain changes that were made on the server side since the data was initially pulled, you have to pull the same result set again (after pushing your changes so that the pulled data contains your changes). RDA does not have any concurrency conflict detection or resolution. So, if you use it with an application on which multiple clients can modify different records at the same time, the last client to write changes to a record could overwrite recent changes by another client (last-in-wins).

The OCS synchronization framework, when released, will allow you to choose between database-to-database synchronization capabilities similar to both RDA and merge replication, or a more service-oriented approach in which you plug your own service endpoints into the synchronization pipeline. The former approach is simpler to set up because it requires the least custom code. The latter approach is a little more complex but enables you to insert business logic between the client end and the server, and also enables you to target other data sources that could be wrapped in an appropriate synchronization service. Figure 1 shows the architecture of the new OCS synchronization framework.

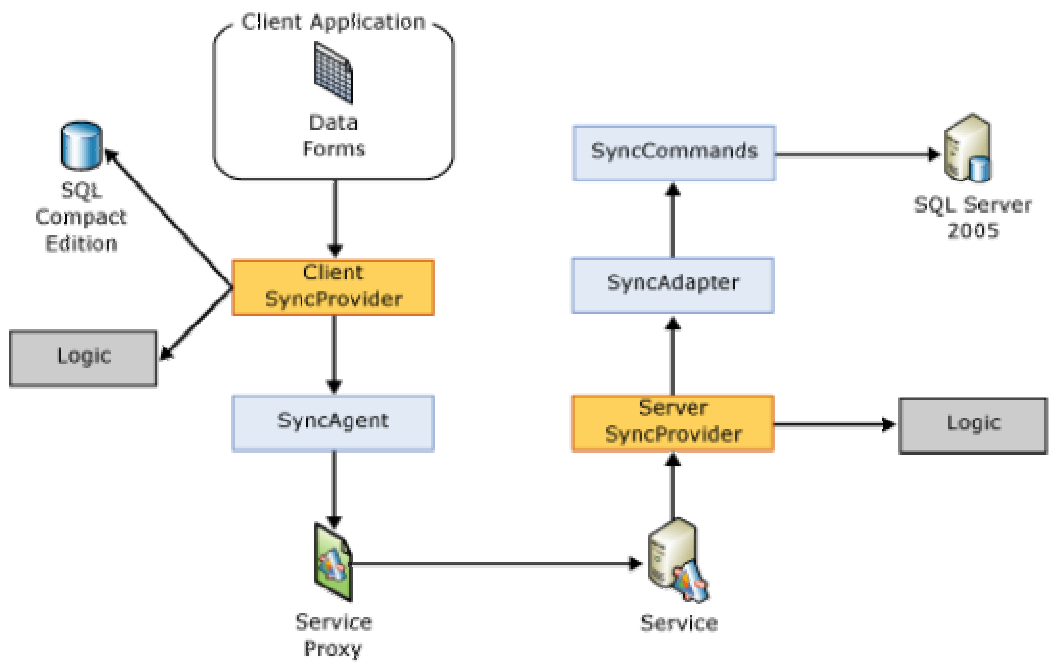

Figure1. OCS synchronization framework

The OCS framework provides an out-of-the-box solution for service-oriented data synchronization, as well as numerous extensibility points to allow more explicit control of the synchronization process. The client calls through a synchronization provider to perform synchronization of data. The provider can call out to custom logic, such as validation logic, if needed. The synchronization provider then calls through a synchronization agent to the synchronization service on the server. The service exposed can be one provided with the framework or a custom service you expose. The service calls through a synchronization provider on the receiving side, which can also call out to custom logic. The synchronization gets dispatched through an appropriate synchronization adapter and its associated database commands. The adapters and commands follow a similar architecture to the TableAdapters associated with ADO.NET 2.0 typed datasets.

SSCE Concurrency

SSCE allows multiple connections to the same database (.sdf file) from the same application or even multiple applications on the same computer. This gives you more freedom to structure your application as needed, such as allowing the user to continue to interact with data while performing synchronization with a back-end database, or to have multiple applications on the same machine share an SSCE data store. Transactional concurrency locks are made by the database engine to prevent concurrent connections from accessing

the same records at the same time. The technical limit on concurrent connections for a single database is 256, but 70-80 is a better practical limit from a performance perspective.

SSCE Security

Role-based security and permissions are not supported in SSCE. SSCE databases are intended to be used as a simple data storage mechanism for local data storage and access on a client computer, or for simple persistent data caching on an application server. As a result, role-based security and permissions at a table level were not included in the interest of keeping the database engine as small and fast as possible. Access control to the database as a whole can be enforced through password-protected access. Protection of the data while it is stored on disk can be done easily by encrypting the database.

SSCE Unmanaged Code Access

SSCE allows you to easily access the data store from unmanaged applications, as well. SSCE includes an OLE DB provider so that applications using Microsoft Access, Microsoft Visual Basic 6, C++, or other forms of unmanaged code can still access and use data that has been placed into a SSCE database. This can help with migration from unmanaged applications and legacy data stores to incrementally move forward to SSCE and managed code, for example.

SSCE Performance

SSCE performance is very good for all of the usage scenarios described in this document. If you compare SSCE performance with SSE, EDB, or Microsoft Access, there is no clear winner across all scenarios. For each combination of query size, storage pattern, and other variables, the performance of one product may look better than the other, but a few trends can be consistently measured.

If you compare SSCE with SSE, several patterns emerge from performance testing. Transactions are slightly faster to begin and commit in SSCE than in SSE, but only by a fraction of a percentage (20 percent). SSCE is faster than SSE for parameterized queries that insert large numbers of rows (> 100), but SSE is faster for single row or small numbers of row inserts. SSCE is about twice as slow as SSE for a single row select and is comparable for 100 rows or more, with the connected model of SqlCeResultSet. However, with the SqlCeConnection API, the first open and the last close are significantly slower in SSCE than SSE (> 1000 times in some cases) owing to the significant overhead of starting and shutting down the storage engine. An optimization for better performance is to open a

SqlCeConnection prior to transactions and close it at the very end. In this case, performance is comparable to the SqlCeResultSet API.

If you compare SSCE performance with EDB on a device, EDB can be significantly faster (100 percent or more) for certain scenarios such as determining database size, and doing inserts or updates. However, most search and query operations favor SSCE by 20-30 percent or are roughly equal in performance.

Architecting Solutions with SSCE

To understand how to integrate SSCE into your application architecture, it is best to discuss its use in the context of the four application types summarized earlier in this white paper: field force applications, personal information management (PIM) applications, small-scale Web client applications, and application server caching.

Field Force Applications

Field force applications (FFAs) are typically client applications designed to run on a mobile device, Tablet PC, or portable computer as part of a larger distributed system that consists of multiple tiers, services, databases, and other architectural elements. FFAs usually share one or more of the following attributes:

- They allow the user to perform their job functions while disconnected from the back-end network—on-site at a client location, on the road, in an airport, or from home. FFAs are usually designed for occasional connectivity, meaning that when users are running the client application, they do not need to have a network connection of any kind.
- FFAs often involve multiple clients that can concurrently access and use data from the back-end database, both in a connected and disconnected mode.
- FFAs must be able to replicate data from the back-end database to the client databases for offline support. They also need to be able to replicate modified, added, or deleted data records from the client to the server when the application is able to connect to the network.
- FFAs must be ClickOnce deployable, allowing frequent updates to functionality, as well as to the database used for local storage.

Examples of FFAs include:

- **Point-of-contact (POC) sales applications for retail, insurance, real estate, financial management, and many other industries.**
- **Customer Resource Management (CRM) applications with extended reach for distributed teams through mobile devices, portable computers, and Tablet PCs.**
- **Healthcare provider applications that provide medical record history access, provider-patient interaction record keeping, ordering of prescription medications, labs and diagnostics, and data collection from measurement devices.**
- **Inventory management applications for warehouses, shipping departments, business office management services, janitorial supplies, and other applications involving recording and tracking item counts and available at field locations.**
- **Data collection and measurement applications such as utility companies, telecommunications, polling, census, and voting applications.**

FFA solution architectures can vary widely depending on the number and type of client platforms and the scale of the application. A typical architecture is depicted in Figure 2. FFAs usually consist of multiple clients involving either PDA/Phone platforms, Tablet PCs, portable computers, or a mix of these. They usually connect to the back-end services through secured Web services or virtual private network (VPN) connections so that they can synchronize with back-end data from any available Internet connection, instead of having to return to the home network to do so. The front end exposed to them is often placed in a perimeter network (also known as *DMZ*, *demilitarized zone*, and *screened subnet*) to limit the number and type of ports opened from it, and the connections into the back-end network are limited. For larger scale applications, the perimeter network Web server may be connected to an application server where business logic and data access components run, and that application server talks directly to the back-end database.

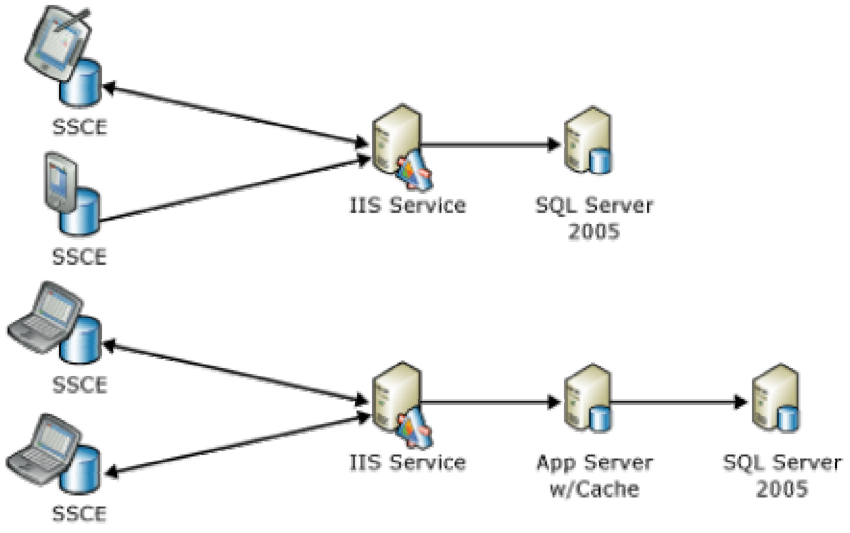

Figure2. Field force application architecture

For data synchronization purposes, a more direct route is often taken to limit the complexity of the application. When using RDA or merge replication, the synchronization architecture looks more like Figure 3. In this case, the client applications connect through HTTP to the back-end database, with Internet Information Services (IIS) exposing the HTTP endpoint through which synchronization communications are done.

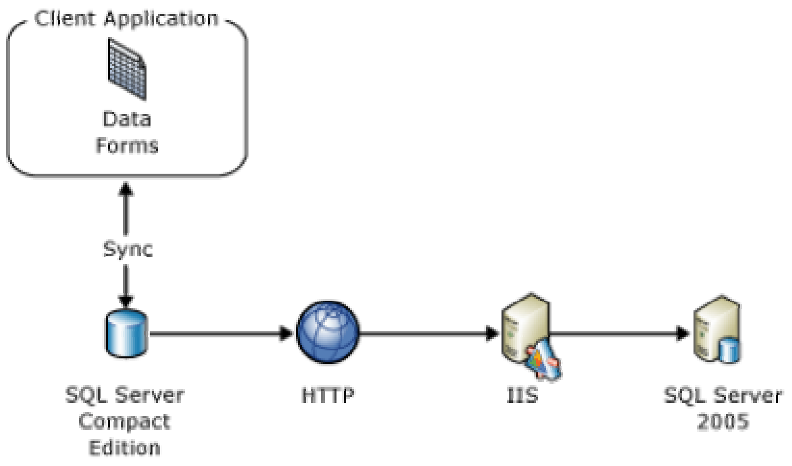

Figure3. Field force synchronization architecture

To perform data access on the client side, normal disconnected ADO.NET data access approaches can be used, such as using typed datasets and their associated table adapters to read and write data from a local SSCE Northwind data store as shown in the following code sample. As you can see, the typed dataset abstracts away all the details of the underlying store, and this data consumer code is no different than it would be if the data were in SQL Server, SSE, or another data store that is accessed through a supported managed ADO.NET provider. Inside the CustomersTableAdapter, SqlCeConnection and

SqlCeCommand objects are defined by the Visual Studio designer and they do all the work of retrieving and updating the data in the SSCE database using normal ADO.NET data access patterns.

Code sample: Read/Write Data Access with Typed DataSets

```
private NorthwindDataSet LoadDataSet()
{
    NorthwindDataSet nwData = new NorthwindDataSet();
    CustomersTableAdapter adapter = new CustomersTableAdapter();
    adapter.Fill(nwData.Customers);
    return nwData;
}

private void SaveChanges(NorthwindDataSet nwData)
{
    CustomersTableAdapter adapter = new CustomersTableAdapter();
    adapter.Update(nwData);
}
```

The SqlCeResultSet also allows for a connected data access model, as shown in the next code sample. In this method, a SqlCeConnection is created, along with a SqlCeCommand to retrieve a row of data (although it works with multi-row result sets as well). We then open the connection, execute the command to retrieve the SqlCeResultSet, and do a data record update through it. In this simple example, the connection is closed through the using block, but normally you would use a SqlCeResultSet if you were going to hold onto it spanning a longer scope of code, and wanted to be able to read and write to the database without having to open and close the connection. In that case, you would not close the connection immediately after using the result set; you would leave it open, and close the SqlCeResultSet when you were done using it.

Code sample: Read/Write Data Access with SqlCeResultSet

```
private void UpdateCompanyName(string custId, string companyName)

{

    // Set up connection, just need path to sdf file

    SqlCeConnection conn =

        new SqlCeConnection(

           @"Data Source ='.\Northwind.sdf'");
```

```
// Set up select query that brings rows into the result set
string selQuery =
   "SELECT * FROM Customers WHERE [Customer ID] = @CustID";
SqlCeCommand getCustCmd = new SqlCeCommand(selQuery, conn);
getCustCmd.Parameters.Add("@CustID", custId);

using (conn)
{
    conn.Open();
    // Pull the row into the result set,
    // using options to allow connected, random, read-write access
    SqlCeResultSet resultSet =
        getCustCmd.ExecuteResultSet(
           ResultSetOptions.Scrollable |
           ResultSetOptions.Updatable);
    // Move to first record
    if (resultSet.ReadFirst())
    {
        // Get the column position
        int ordinal = resultSet.GetOrdinal("Company Name");
        // Set the value
        resultSet.SetString(ordinal, companyName);
        // persist
        resultSet.Update();
    }
```

```
        else // No match

        {

            throw new ArgumentException("Customer not found");

        }

    }

}
```

Personal Information Management Applications

PIM applications are simple data applications that run against local data stores on mobile platforms for easy access and storage of personal information. PIM applications usually share one or more of the following attributes:

- **Run on mobile platforms, including phones, PDAs, Tablet PCs, or portable computers.**
- **Provide simple data entry and lookup against local data stores for application data.**
- **Require a minimum footprint for availability on mobile devices.**

Examples of PIM applications include the following:

- **E-mail, instant messaging, contacts, tasks, notes entry, storage, and retrieval.**
- **News reader, RSS aggregation.**
- **Calorie counting, time management, exercise logging.**
- **Language tutor, vocabulary coach, dictionary, thesaurus.**
- **Shopping list, CD/DVD collection.**

The solution architecture for PIM applications is very simple, as shown in Figure 4. It follows a classic client-server design in which the server is a local database engine on the device or portable computer. EDB could be used for mobile devices, but SSCE makes a simple-to-use, powerful, yet lightweight option for any Windows platform, including mobile devices. For example, SSCE offers richer query support than EDB, and it is easier to develop applications using SSCE in managed code.

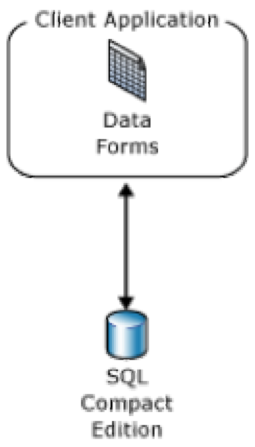

Figure4. PIM application architecture

A PIM application that uses SSCE for a database engine is likely to use the SqlCeResultSet API to make data retrieval and storage as simple as possible. The code for this would look very much like the second code sample in the previous section.

Small-Scale Web Applications

For simple Web sites with low traffic expectations, there is no need to buy expensive server software or to pay for higher cost hosting options that involve a database instance just to have a dynamic, data-driven Web site. Both SSCE and SSE provide a simple and free option for storing dynamic data that can be used to render content on Web pages. Whether the Web site uses ASP.NET or some other Web technology capable of doing data access through an OLE DB provider, the data that feeds the site's pages can be served up by SSCE or SSE.

Web applications that are suitable for SSCE or SSE are those that have a small user base—the total number of users may be large, but the number of concurrent requests per second should be fairly small.

Examples of Web applications that might be candidates for SSE or SSCE include:

- **Hobby/club sites with a small member base.**
- **Small user group organizations.**
- **Internal Web applications for small companies.**

Unless the expected utilization of the Web application is extremely low with little potential to scale to larger numbers of users, SSE is generally a better option for small-scale Web applications than SSCE. If the application remains a single-process Web application with a

very low number of requests per hour (< 300-500 requests per hour, for example), either SSCE or SSE should be able to handle the throughput required.

However, as the complexity of the application grows and/or the number of requests per second increase, SSE offers better options for incrementally scaling the application to meet increased demand. As described earlier in this paper, SSE supports stored procedures, functions, and other enterprise-level features that can provide a more layered and decoupled architecture for complex applications. Additionally, because SSE runs as a service, concurrent access from multiple Web application processes is better supported from the same or even different computers. There are more options for role-based security in SSE, and the upgrade path to a full version of SQL Server is seamless—no change in data access code should be required, because SSE and SQL Server use the same data provider. Also, the file formats for SSE and SQL Server are the same, so the SSE database can just be attached to a full SQL Server instance and it is ready to go. Switching from SSCE to SSE or a full version of SQL Server requires a data migration, as well as re-coding some of the data logic since SSCE uses a different data provider from SSE or SQL Server.

If you choose to use SSCE for Web applications, some configuration is required to allow SSCE to be hosted in the Web application process.

Caching Applications

Caching applications can be client-side applications, or application server applications that need to maintain a local data cache of lookup values or rarely changing data to avoid unnecessary roundtrips to the back-end database. Caching is used to allow the application to retrieve the needed data from the local computer to improve performance and reduce network traffic. Caching application use cases include frequent access to lists of relatively static data records that are used for presentation purposes (drop-down selection or reference lists), or for application server lookups in business logic.

For the fastest data retrieval, data can be cached in memory and returned directly when the cached values are needed. However, you may not want to or be able to refresh the cache from the server each time the application restarts, so a local data store for the cached values may be needed. Additionally, if the lists of values you are caching are very large, such as a product catalog, you may not want to hold the entire catalog in application memory as the items in the collection are only accessed sporadically. For either of these reasons, using SSCE as a local data cache storage location may be a good choice for your architecture.

Cached data needs to be initialized from the server the first time the application is run, and cached data often has an expiration date or time associated with it when the cache must be refreshed from the source. The time-out associated with the cache is determined by the application requirements, how frequently the data changes, and how much latency in the cached data can be tolerated by the application and the users.

When designing a data cache into an application, you must choose between a passive or active data caching strategy. With *passive caching*, the application tries to obtain the needed data from the local data cache. If the data is not present, application logic retrieves the data from the server and stores the results in the cache for subsequent retrievals. When the cache becomes invalid due to a time-out or other criteria, the application must refresh the cache from the source. With *active caching*, the cache is actively kept up to date based on some criteria, such as a timed refresh from the server. When an active caching strategy is used, the application code is simpler because it can always simply refer to the local cache for its data. However, additional code is required to initialize the active cache and keep it up to date.

For a caching application, you might want to do two-level caching. You might want to cache the data in a persistent store, such as SSCE, to allow the cache to be resident on the local machine to survive application restarts or to avoid having to load all of the cached data into memory. However, you may also cache some of the data from the persistent cache in memory to provide the fastest possible retrieval time when the data is needed (for example, to drive auto-completion logic on text box or drop-down list input controls). For these two levels of caching, you could have the same or different caching strategies (active versus passive).

Figure 5 shows a sample caching application architecture, with caches existing in both the client application and the middle-tier application server. The client application caches data both in the persistent SSCE store and in memory because memory consumption is usually less of a concern in a client application. The application server cache stores large lookup lists such as a product catalog persistently to avoid consuming memory for scalability reasons, but avoids the performance impact of making roundtrips to the database server whenever a product catalog lookup is required for something like locating all products within a certain price range. The main difference between a caching application and one that synchronizes (such as a field force application) is that data is only pulled from the server into the local data cache in a caching application. With synchronization, the data from the cache is also synchronized back to the back-end database after edits have been made.

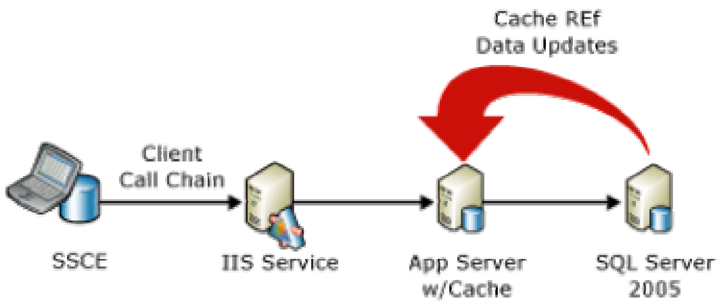

Figure5. Caching application architecture

Conclusion

Many applications need a reliable, transactional, persistent, efficient data store but do not necessarily require the full range of capabilities offered by SQL Server 2005. SSCE provides a free, powerful, and easy-to-use database engine that can be used in a wide range of application scenarios and architectures. It can be used for field force applications, personal information management applications, small Web applications, or application data caching. It provides a number of current and emerging data synchronization options to allow it to be part of a larger distributed application architecture involving back-end SQL Server 2005 databases. SSCE should be one of the database technologies considered for any application needing a lightweight relational data store.

***~~~ End of Article ~~~***