TIERED ARCHITECTURES AND DATA ACCESS LAYERS

The importance of a well-defined and implemented Data Access Layer (DAL) for any environment impacts multiple areas of IT domains namely Database, Infrastructure, Network, Security and Application/Development. Important influencers for a well-planned DAL are Database Isolation, Database Performance, Standardized Application Endpoints and Security. The complexities surrounding the design and implementation of a DAL are the root-cause for their typical remittance from established Application frameworks. The "who" is responsible for a DAL has consistently been the excuse for not adopting this framework and as of current, there is no marketed Data Access Layer Generation tool that fully creates Stored Procedures, Web Service Interfaces and the framework necessary to support this architecture; until now!

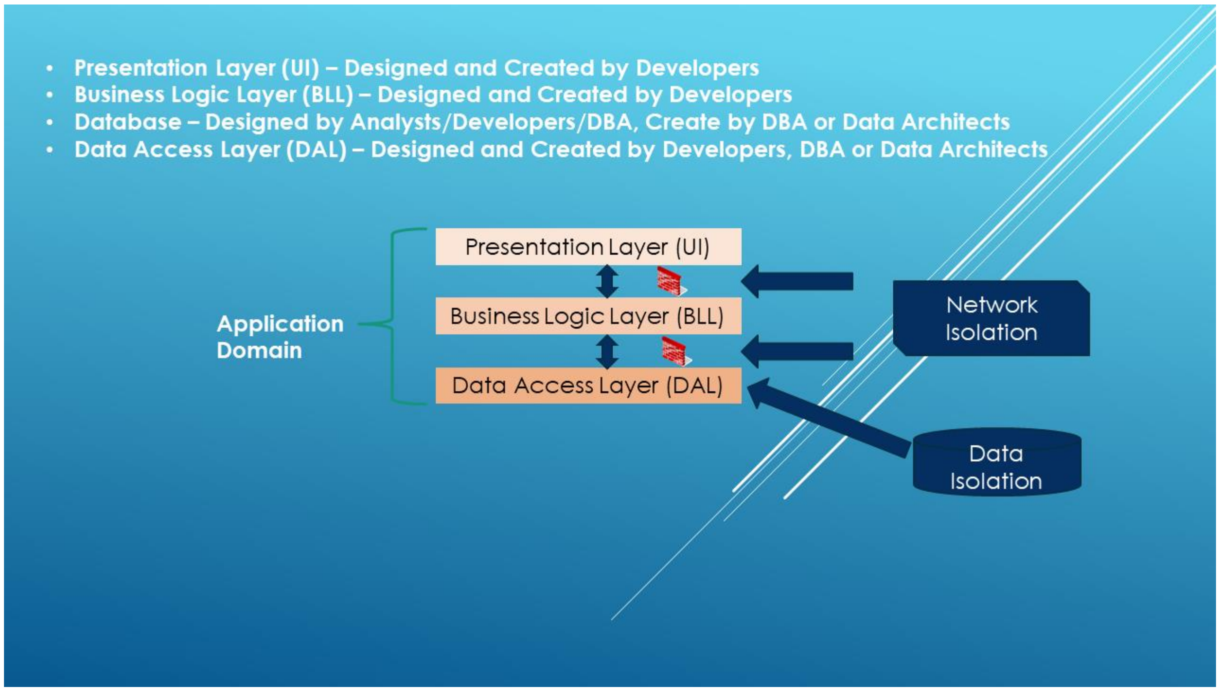
Let us get into this with a review of Tiered Architectures in order to understand why fully Architected Data Access Layers have been seemingly omitted from implementation by organizations from the smallest to the largest enterprises.

BACKGROUND – TIERED ARCHITECTURES

There are a wide variety of definitions for Application Tiered Architectures but simply stated; Tiered Architectures are the delineation of the development by models/domains commonly named Presentation Layer (or User Interface aka UI), Application Layer (or Business Logic Layer - aka BLL), and the Data/Resource Layer (or Data Access Layer aka DAL).

Each area has specific development, security and integration aspects with complexities increasing from the Data layer, through the Business Logic Layer and culminating in the Presentation Layer where the most complex functionality tends to reside.

The image below depicts a Three-Tier Architecture



- Presentation Layer (UI) – Designed and Created by Developers
- Business Logic Layer (BLL) – Designed and Created by Developers
- Database – Designed by Analysts/Developers/DBA, Create by DBA or Data Architects
- Data Access Layer (DAL) – Designed and Created by Developers, DBA or Data Architects

In the development arena, there are numerous variations for developing applications depending on the scope, size, and complexity of the application, the levels of isolation between the ownership of the layers, organizational and security policies and a plethora of other elements all of which impact the overall success, cost and ownership of the final solution.

Before we dive into the next section, let us consider an IT philosophical “baseline” with regards to best practices for Application Development and how “Tiering” should be applied. Starting with data which is commonly comprised of a relational database usually Microsoft’s SQL Server or Oracle’s Database offerings. There are a host of other data repositories but for lack of time and readability, let’s focus on “Relational” database for sake of discussion and the potential for disparity. Databases have multiple features and elements that provide diverse “tools” to store, retrieve, update, delete, secure, audit, log, enforce rules, and too many more verbs to note. Suffice to say, an enterprise database has all the inner workings to allow data any way a person needs it. This may not be the best policy from a “management” or a “security” point of view. Like a car which has an engine and lots of capabilities, misuse by going too fast and not operating in control has its consequences. The same holds true for the utilization of a database; misuse also has its consequences albeit not as catastrophic as driving out of control in a car but nonetheless, misuse can be costly and catastrophic to an organization.

Which those analogies, let us focus on “data” and how it “should” be managed, again not going too deep into database administration and control but more of a “best practice” point of view. In a database, tables contain fields and some of these fields makeup “keys” which allow data to be unique. Other elements known as Indexes are assigned to field(s) and these indexes allow the database to “manage” efficient methods to store and retrieve data. In short, properly creating tables, assigning key fields, and then defining indexes for field(s) initiates the baseline for a properly defined database. One additional and pertinent element is a “Foreign Key” which is used to control tables in a “parent/child” relationship. Simply stated, a parent “record” with a “Key” is required to support the creation of a child record in another table that includes the Parent “Key” as one of its fields. If the parent exists, the child table can have a record created that contains the Parent key. In the event a “parent” record needs to be delete; it is critical that there are no child records that exist or a violation occurs. A child record can be updated or delete, if necessary, without regards but the parent “Key” values cannot change, and the Parent record cannot be deleted until all the children records have been removed. This hierarchy is relevant to database designs, and to the scope of Application Development and ultimately, as part of “Business Rules” which govern how the Application will be designed and developed. In a General Ledger, you cannot remove a Master group aka Account, and Sub-Account, if transactions still exist for the Account and Sub-Account. You cannot leave a restaurant until you pay for the meal; Checking Account Debit and Credits follow this relationship as do much of everyday interactions with businesses.

In summary we have a high-level scope of a database, and some of the minimum elements needed to define and control data. The next step is where the Data Access Layer tier begins to delineate. The common best practice to “secure” and manage a database falls into two distinct areas; proper scripting or queries and the permission or “role” assigned to execute these scripts or queries. Database access by an application generally falls into two methods, Dynamic or Structured aka “Static” access whereas dynamic is when the application and/or ancillary tools directly query the database. Dynamic is the “quick and dirty” method of data access as it requires that the database be open to direct manipulation. There needs to be an account with appropriate privileges to perform creation, retrieval, updates, deletions of data, aka CRUD.

This is the first point of ‘issue’ as the database can be compromised due to the unconstrained nature of dynamic access methods. Next comes the lack of adherence to the “keys” and “indexes” as noted above. Consider multiple, very large tables with hundreds of thousands of rows, (ok large can be hundreds of millions of records) and developers have created “queries” that select data in these tables, but selection logic includes fields that are not primary keys or indexed (or a part of them). In this case, the database has to perform what is known as a full table-scan which wreaks havoc on the database environment, impacts the Server supporting the database and finally causes extensive Read I/O to the storage array. Now, compound that with hundreds of people using the application with numerous other similar “queries” such as this one and you have a situation where performance impacts the applications usability. Going further, now you have the infrastructure team becoming involved; oops; jumped too far ahead; this is a small company. So, the developer has to try to figure out why the application is unbearably slow. The resolution is to spend lots of time and resource searching for answers (try googling) and in the end, the company wastes time and money as the application and the database were poorly designed and can only be addressed by spending vital funds unnecessarily on more hardware to resolve the problem. Adherence to well-structured methodologies and architectures would have allowed the application and the database to be developed and to operate efficiently.

What exactly is “Structured” data access? It is the implementation of well-defined database Stored Procedures, and Views as the primary method to access data by an application. Stored Procedures are the most efficient method to access and control data and this concept has been noted and published for decades by prominent professionals, specialists, and experts in the fields of Application and Database design and utilizations. The execution of Stored Procedures actually become part of a Database “Query and Execution plans” (see image below; from Microsoft) as they are in effect “compiled” by the database for optimization. The role or privilege necessary to execute them is known as “EXEC” and this only enables the execution of a Stored Procedure. With this level of control and access, a user or external program cannot make unstructured “Dynamic” queries against the database and overall database performance can be maintained and monitored efficiently.

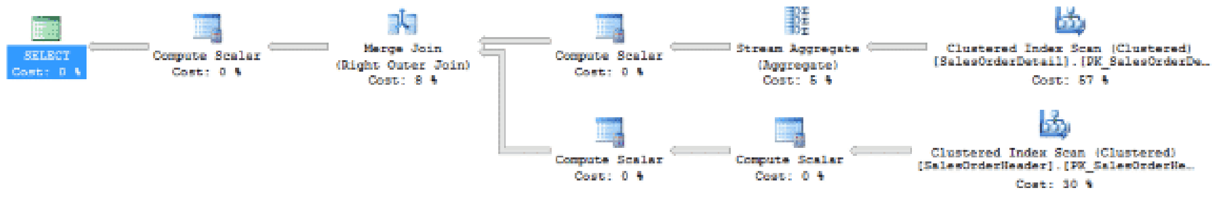


Figure 1 SQL Execution Plan

This concept is at the heart of a Data Access Layer whether this be via direct calls to Stored Procedures by Applications or via Web Services. There are advantages of both methods depending on the environment and the necessity to maintain strict separation between the Data Access Layer and the Business Logic Layer. If isolation is not a consideration and the DAL and BLL can be co-located in the same or adjacent systems, then direct Database access to execute Stored Procedures is adequate. When isolation and access are necessary such as when data needs to be accessed from multiple applications, both internal and external to the environment, then Web Services at the perimeter for exposing the Data Access Layer is the best solution.

THE DATA ACCESS LAYER ADOPTION INFORMATION GAP

Consider 3 different organizations, small, medium, and large, their Software Development Life Cycle or SDLC in regards to Tiered Architectures, and the impact and cost of ownership for implementing this framework.

Let us start small and work to large entities so we can see how and why the adoption, implementation, managing and supporting Tiered Architectures changes with the size and complexity of an organization.

SMALL ORGANIZATIONS

First focus is small businesses which understandably tend to have smaller IT staffs and depending on the scope of their projects, their solutions may be either a Commercial Off-The Shelf (or COTS) which may be integrated with other internally developed business systems or complexly designed and developed internally. In small IT departments, specialists perform multiple duties across the Application Tiers, and this includes database design and management, the entire data logic realm, business logic definition, and the presentation layer design and development, and finally the application and data deployment. In essence, a “developer” may have full ownership of the application from inception through deployment and potentially managing security along with ownership of portions of the infrastructure. In many cases, workloads may be split but each member of the team usually has full knowledge of the SDLC and can maintain and support entire solutions.

In small organizations, using best practices is not a primary focus as there is little to no peer review and there is commonly no requirement to maintain high levels of structure or standards. Small team developers tend to use the Rapid Applications

Development (RAD) methodology of project management which translates to using integrated tools to access and control data. In the Microsoft .Net/Visual Studio domain, this equates to utilizing Entity Framework, DB Factory, Active Data Objects (ADO), or similar data access/control tools. All of these tools require extended privilege access to the database, at a minimum, to "Create, Select, Update, and Delete" data. In the face of necessity, most developers are provided "Full" or Database Owner "DBO" access which allows the developers/application to have full, unimpeded access and control of the data repository. Small organizations tend to implement applications that are isolated and there may be minimal requirement for external access, so this level of authorization satisfies the requirements of the developers. If this model is adopted, performance of the database is not generally a concern or a focus; the mantra is just "get the data" and until performance becomes an issue, it is not considered. The sooner the application is designed, developed, and deployed, the better for the organization. Developing proper Stored Procedures and Views and the rigors of testing and implementing them is considered by developers, in most cases, as duplicative since the they can simply use dynamic SQL statements to get the data and the call can be embedded in the code. Testing and hardening is secondary especially where the is minimal opportunity for peer-reviews, oversight as mandates to conform to development best-practices.

Why write Stored Procedures and Views and then maintain both the code and the SQL artifacts?

Let us revisit the "SQL Plan", Performance issues, Isolation concerns, Security controls, and the best practices for Stored Procedures and Views as noted above. One reason for their absence is that perhaps these fundamentals were never taught as a cohesive set of interdependent methodologies to developers in the first place? 4 large Educational institutions I attended, all of which offer Information Technology programming as part of Information Science degrees, did "NOT" teach integrating Stored Procedures with Application Development nor any of the above principles/methodologies. These institutions did not explain or teach the full Software Development Life Cycle and its nuances which is at the heart of why DAL's are rarely implemented and why they are-all-too often omitted from an application.

Even the Data Structures courses (useless name for a course) only briefly touched on Stored Procedures and Views. The emphasis here was the basics of writing SQL statements, table creation and their relationships and basic methods for data access via dynamic T-SQL/P-SQL scripting in the high-level language. While it is pertinent that education institutes have limited time and resources for passing on knowledge, the relevance and importance of a Data Access Layer should be as fundamental as developing a User Interface.

To validate this premise, ask one of your developers or a DBA if they were formally trained on implementing Stored Procedures and Views from a "programming perspective" as part of their core programming courses; emphasize not as part of their Data Structures courses. Also inquire as to where they acquired their acumen for data access, data design and data security.

MEDIUM ORGANIZATIONS

Moving up to the medium sized organization, we have a defined IT team that usually has separation of ownership and responsibly between developers, database administrators and infrastructure. In these organizations, there is often delineation in the development domain that segments Presentation, Business Logic and Data Access. Multiple members of this domain work together to design, develop and deploy solutions that range from rudimentary to complex and may have data that is derived from multiple systems and sources. This scope applies to smaller organizations but typically become more complex as organizations expand and points of data and operations are diverse and separated geographically.

At this point, data access tends to be more rigid as peer oversight, commonly from an IT Steering committee or an IT Director/CIO implement policies and standards to control access and dissemination of data. Event with medium sized organizations, there is a mix of adopted and implemented standards and ad-hoc or "leave it up to the developer" to implement their methodology to access and control data. Medium sized organizations tend who allow ad-hoc data access and control methods tend to have more issues with data access performance and in many cases, resolve the problem with hardware; “more is better” is the typical stance of the infrastructure team. With medium sized organizations we also have more points of access to data from external systems and the necessity to coalesce data for reporting, M&A activity and for Supply-Chain opportunities. The more expansive an organization becomes the larger the data farm and the increase in adopting best practices, especially with regards to data access.

We are now at a point that simply allowing every application to have the ability to access data “anyway they want it” becomes unsustainable and eventually too costly to maintain and ignore. Once Ad hoc is adopted it becomes increasingly difficult to revert to a standardized approach especially with regards to development and DBA resources to modify existing applications and databases. Once a system goes live and multiple external systems and users utilize that data, the more complex and difficult it becomes to change the environment. The SDLC becomes unwieldly and unmanageable as “stability” and “efficiency” is the common avenue. An effort to change any operational application is analogous to the infamous “rewrite” which is a term avoided in IT circles like the plague. Management’s stance is typically “*If it is working, don’t touch it; otherwise you own it and will be blamed when it fails; and it will fail if you touch it*”. Sounds more like “FUD” – Fear, Uncertainty and Doubt. In addition, if staff rollover has occurred and key developers are no longer available and as per usually, there is little to no code documentation (there rarely is), then the code is “cemented”, as-is for eternity…

The crux of the problem with application development and the lack of adopting and implementing best practices for a DAL starts with honing the appropriate skills not just for writing code and UI scripting but also to manage and control data. All too often, the developer is poised to design and develop their data repository, typically in a “Sandbox” or a copy of a “like” production environment whereby the developer simply uses embedded frameworks to access and control data. And why not; the features are provided by the IDE! On the other hand, writing, testing, and implementing Stored Procedures is time consuming and considering most developer’s acumen, an unnecessary effort. Many developers have little to no experience with SQL script development nor do they have an in-depth knowledge for the usage of Indexes, Keys, Foreign Keys, or best practices for SQL syntax. All too often, I have heard the statement “*Why waste time developing Stored Procedures and Views when we can have full control with 1 framework object: DB Factory, Entity Framework or ADO*”.

With medium sized organizations, database administrators are busy handling request for sandboxes, deployment issues, back/recovery issues, contingency planning and a slew of other issues, including meetings, training, evaluating new tools, provisioning new database, adding privileges and accounts, etc., etc., etc... This is the second problem with the lack of adoption of a DAL; DBA’s are typically not engaged with the application team in the development of the solution from a data perspective as the developers often have a majority of the knowledge concerning what data they are collecting/manipulating, and in the end, the developer simply maintain their own data sandbox and create/update/control a local application data repository.

Once the application is ready for deployment up the SDLC stack, the DBA generally copies the tables, if not the entire database up to the next level (SYSTEST). How the application works, what indexes (if any) are created, observance to Foreign Keys or even a rudimentary overview of how the database was designed is not the primary focus of a DBA; just move the data up to the next level for testing.

A key problem starts here as the SQL Plan is often omis from any review by the DBA or Data Architects as in most instances, there is no plan because there are no SQL Scripts; all the queries are dynamic, embedded SQL Statements that are infused with the application code. This leaves the DBA with 2 options, (1) review the code - wait, DBA’s are not programmers and there could be hundreds of thousands of lines of code, or (2) trust that the programmers have developed proper SQL statements, handle any issues that arise with database monitoring tools, and address problems if/when the application fails or exhibits performance issues; all after the fact….!!! This is the infamous; it is not my responsibility in an IT parlance.

"Programmers do not design/manage databases and DBA's don't program"; Everywhere I go, this is the mantra!

The Business Philosophy: (<u>Medium/Large businesses) Isolation and Segmentation of Roles and Responsibilities by Departments is fundamental for Management to Implement and Manage a Hierarchy (MBA 101)</u>

LARGE ORGANIZATIONS

Lastly, we examine large organizations which often have large IT departments, and if the organization is diverse, can have multiple IT Departments dispersed geographically. Consider GE, Alcatel, Tyco Laboratories, UTC, Siemens, Microsoft, Boeing, Air Bus to mention a few. Having worked for the first four, and for more than a decade for the first 3, observations of their IT departments confirmed the effect of both small and medium sized organizations with regards to the standards and approaches to application development and data base utilization. Large organizations often have multiple dispersed systems and yet many of them need to interact and share information constantly, several on a real-time basis. This is most common with financial, inventory, sales, and production systems.

Large entities have complex IT departments with multiple layers of domain segmentation and even further levels of ownership and specificity within the technology areas. With GE, the database team had minimal interaction with the application teams; it was purely what database do you need and where do you need it. You want it, you get it and we will back it up and create accounts for you. Aside from that, you are on your own. Tyco and Alcatel followed that mantra. The same philosophy was readily present with both the State of Connecticut's Judicial Branch and Santa Clara County California. DBA's and Developers were akin to Oil and Vinegar; they mix when shaken but when left alone; the separate.

The issue with large organizations is a combination of attrition and acquisition and these are not mutually exclusive as will be readily apparent. First, large organization grow by a combination of expansion and through mergers and acquisition (M&A). With M&A activity, systems from the merged or acquired company are often integrated with the existing technology solutions of the parent entity as all too often, it is more complex and conflict ridden to attempt to depreciate the acquired/merged company's system than it is to integrate them. Integration simply stated links data between systems. Again, the "*quick and dirty*" comes around and the fastest method to integrate is to use either development tools that are prevalent in the parent company's IT toolbox or a data integration tool such as Informatica, CONNX, or simply use SQL/Oracle synchronization technologies. The later model simply extends the remote data onto local databases for access by the parent entities systems. The problem is that there is no business logic, data controls, or adherence to how to manage and respect the remote data schema. It is often just treated as a "data dump" and in some cases, is exactly that, a refuse dump of data!

So, the large company has multiple data repositories, some "clean" and others, not so clean. Now we focus on development and again, we have deeper levels of isolation between the application teams and the database teams and even further down the road is the infrastructure and Security groups. The application teams in large entities are often compartmentalized and each group or cell usually has "ownership" of specific solutions or sets of solutions such as Finance, Sales/Marketing, Production Control, Inventory/Procurement and lastly Reporting which is a group that actively uses data from all the other business domain. With large organizations comes the architecture to integrate and share data between applications, organization and external partners as part of the supply chain (vendors, customers, partners, etc.). Solutions that need external exposure often have a combination of methods to access data including direct access (dynamic SQL) or Web Service. Externally access, as of recent is usually via Web Service but in older systems, may be supported by EDI or even pure TCP/IP communications both of which may have implementations of direct access or Web Service frameworks.

The problem (yes there is one here also) is that the large organizations have a melting-pot of technologies, methodologies, policies, standards, and practices that constrain adopting and implementing a foundation for consistent data control and access. Changing an existing system is expensive; adopting standards mid-stream in a project is time consuming, hence expensive; acquired systems that support external/merged/acquired entities requires resources, time, testing, training and again; is expensive.

The base of the problem is the dubious problem of re-writing applications with the focus on the best practice of utilizing Stored Procedures and Views and implementing Web Services for all data access while limiting control to a database to only a “EXEC” privilege.

SUMMATION

THE WHO, WHAT, WHY, HOW, AND WHEN OF DATA ACCESS LAYERS

THE WHO

Developers do not want to write stored procedures, nor have they been trained to do so, and DBA’s are too busy to write Stored Procedures and View and the have limited scope as to the Applications entire function, particularly with regards to handling of data. DBA’s typically have a full-schedule and in medium/large organizations, they are often juggling many activities simultaneously and they are a critical, limited resource in any organization.

This leaves an unresolved question that even in today’s IT domains, who or more succinctly, which IT department is responsible for the design, development, implementation, and management (SDLC) of a Data Access Layer? In most organizations of these sizes, there are two departments, Applications and Database; who owns it?

THE WHAT

Consider a simple formula for the scope of creating a Data Access Layer. Let’s apply a schema consisting of the following:

10 tables, each with 10 fields, and each table has 2 indexes and a primary key. We will implement both SOAP and REST endpoints and we will generate all the Stored Procedures for CRUD and Index access.

(1) We will require stored procedures for core CRUD (Create, Retrieve, Update and Delete). These functions equate to 10 tables * 4 (CRUD) functions = 40 stored procedures as a starting point.
(2) Each table has 2 indexes and these are most often used to Retrieve (Select) data but they can also be used to Update and Delete data so we’ll use 3 (RUD) functions which equates to 10 tables * 3 RUD functions * 2 indexes/table or 60 additional stored procedures which totals to 100 stored procedures.
(3) To support the Web Service Interfaces through their respective Services and to access the Database (via EXEC privilege) we need Class functions; this is a 1:1 relationship between each stored procedure and the Class function and this equates to 100 Class functions (VB.NET)
(4) For each Class function, we are exposing a SOAP web service (Web Service) which is also a 1:1 relationship, one endpoint to a Class function or 100 SOAP web service routines.
(5) For each SOAP web service, we require an Interface to expose the endpoint and this is linked to the web service and is also 1:1 or 100 Interface routines for each web service. (A Web Service endpoint typically has Interface and Service code)
(6) We also need REST Web Service, so we add another 100 Web Service routines plus 100 Interface routines which is another 200 routines
(7) Now for the Grand Total: 100 Stored Procedures, 200 SOAP and 200 REST Interface/Service routines, and lastly 100 Class functions to support the REST and SOAP Service routines for a total of 500 Functions/routines and 100 Stored

Procedures. These elements are created in 2 Interface files (SOAP/REST), 2 Service files (SOAP/REST), and 1 Class file. Stored Procedures are automatically generated in the database, so no files are generated for them.

a. As a sample measure of lines of code for the Services, Class and Stored Procedure, the demo used for developing DAL_GEN consisted of 17 Tables (the ErrorLog table is optionally added). Each table has 1 primary key, 1 Foreign Key was added to 1 Table, Tables have between 2-5 Indexes and there are 46 indexes in the database and finally there are 166 columns. The code totals are as follows:
    i. REST Interface – 961 lines
    ii. REST Service – 2362 lines
    iii. SOAP Interface – 916 lines
    iv. SOAP Services – 2403 lines
    v. Class file – 7926 lines
    vi. Generated Stored Procedures – 10339 lines

b. The DAL code set (REST & SOAP) and the creation of Stored Procedures was generated in 14.3 seconds

THE WHY

Developing Web Service standards that utilize well-defined Stored Procedures and exposes data in a uniform framework is critical to supporting database access, program design and security policies. The level of isolation that is afforded by this framework, while lengthy, allows for consistent data to application access and establish a baseline that promotes ease of support and modifications, addresses security constraints and promotes efficient use of database resources and lastly contributes to the overall network, database, OS and storage performance of the solution.

THE HOW

(1) Have a Developer or a DBA develop the code and stored procedures manually and all nuances that accompany the effort (Testing, Application Integration (lots of applications and lots of additional testing), Promotions, future database changes, documentation, etc.) or

(2) Utilize Konigi's DAL_GEN and automatically generate all the Stored Procedures, and core application Classes, Functions and Interfaces for your Web Service (CRUD, Indexes, Foreign Keys) implementation.
    a. The only activity is to update the Web.Config file (connection, and Endpoints) for the REST and/or SOAP interfaces and add a database connection string.
    b. Compile and deploy the Web Service application and test the connections.

AND WHEN

When you contact us at Konigi we will show you how efficient, effective, and effortless implementing a Data Access Layer can be for your IT department.

Richard Ladendecker