IBM i Enterprise Modernization

Leveraging DB2 for i SQL in a
Data-Centric Design Paradigm with
SDM (Strategic Data Management) and
MDM (Master Data Management) as management objective.

Those within the IBM i community know the platform for its reliability, scalability and for delivering tremendous business value with unusually low operational costs. So why is the system perceived as being "legacy?" Many generalize their thoughts on the green-screen user interface or a lack of effort to publicize the value proposition of IBM i, or both.

Nonetheless, based on extensive research by IBM and others, we strongly believe that the lack of application agility is the number one factor limiting the growth and acceptance of this remarkable platform. In fact, the biggest constraint of application agility is neither the user interface nor the RPG-written applications, it's the fact that we're not using the SQL capabilities in DB2 for IBM i database and data centric design paradigms.

This whitepaper will explain:

1. Why companies should consider modernizing their application's database;
2. The database modernization challenges companies have faced in the past;
3. The benefits of migrating legacy or heritage database objects;
4. Why we should implement data centricity in our applications and make our database "self-aware" and "self-enforcing";
5. The new technology advancements that can be leveraged after database modernization has taken place;
6. The crucial issue of data quality and integrity and the impact of neglect;
7. Why implementing SDM (Strategic Data Management) or MDM (Master Data Management) principles on our database are important.

First and foremost, it is important to ask the following:

- ✓ **What is the cumulative value of the intellectual capital invested in your applications?**
- ✓ **What competitive advantage do your current application systems provide?**
- ✓ **What is the value of the data and information captured and stored in the heritage application database?**
- ✓ **What constraints prevent your business from responding with AGILITY to competitive pressures?**

Without a doubt, there is significant value in your "legacy" or "heritage" applications, and this value can be leveraged relatively easily for dramatic results and benefit to your company. It is entirely feasible (rather simple, actually) to extend the life of these applications while remaining competitive. Our goal is to build a case for database modernization that supports your business goals by defining the value in your heritage application system.

Modernize Your IBM i Software Assets

IT modernization is the process of extending the value of aging platforms and systems into the future by adopting more modern and efficient approaches and technologies. This can be accomplished whilst removing the operational constraints that inhibit growth and agile response to changes in the business environment. However, what should be thought of as an ongoing process of IT system improvement has become known to be a large and seemingly daunting project, because incremental improvements to monolithic code often exhibit "gotchas" along the way. As a result, the importance of system improvement is often forgotten as IT personnel are consistently challenged to 'do more with less.'

Interestingly, modernization has continued to be a top priority for CIOs. The results of a survey conducted by Gartner of over 2,300 CIOs indicates "Legacy Modernization" as being one of the top-10 CIO business and technology priorities for 2012.[1] The importance of modernization has since only increased.

So, why is IBM i applications perceived as legacy?

1. **User Interface/User Experience (UI/UX)** – The user interface is the most visible part of the application that younger CIOs and CFOs see as being the default interface of the IBM i and as such are often quick to pigeon-hole the entire platform as legacy.

2. **The way the "database" presents itself and its contents to the outside world** – In addition, younger programmers and users cannot relate to the 6 – 8 character field names and the 6 – 10 character file names of DB2 for i, which older programmers historically was forced to employ to name database constructs and elements. This prevents the easy extraction of information without technical assistance, to interpret the nomenclature.

3. **Old style monolithic code** – In the early years, developers was often forced to create huge "structured" monolithic programs, as that was the programming model of the time, due to compiler restrictions. As a rule-of-thumb", 80% of the lines of code in these monoliths dealt with database (entity) relationships and database validations. Additionally, there was little-to-no separation of function, which made maintenance increasingly problematic. The way in which this presented itself, was massive maintenance backlogs and frustrated users.

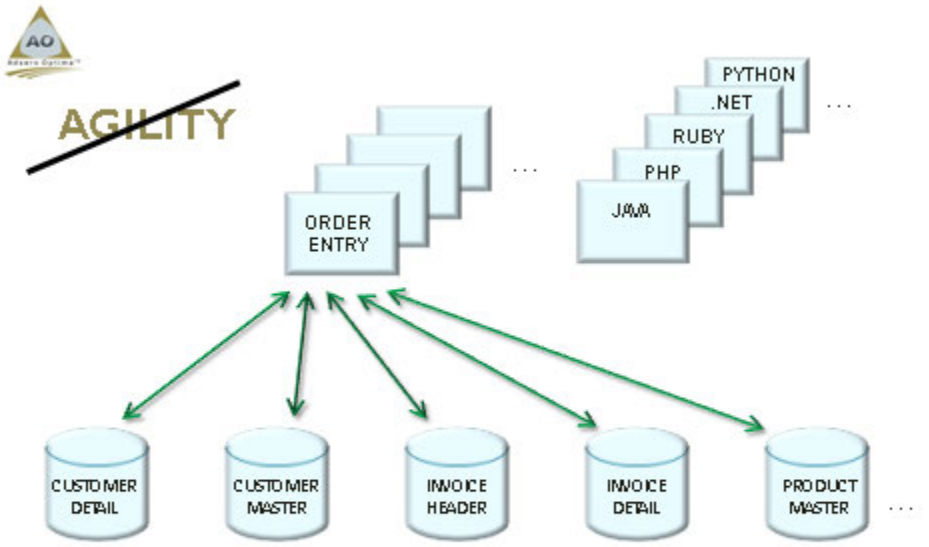
We seldom or, more accurately, never improved our applications by introducing newer coding techniques, leveraging new enhancements to the language and programming model (OPM versus ILE), operating system and database, not to mention improving the maintainability of the applications, because the applications just kept on running. Massive technical debt collected in our applications, due to decades of neglect. Thankfully, the platform (IBM i) was so forgiving, which allowed those applications to keep on functioning.

Now, as IBM shops strive to remain competitive but are facing several challenges: skyrocketing costs are constraining IT budgets; the many years of intellectual property invested in heritage applications need to be retained and leveraged and major advances in analytics, cloud, mobile, and database engine technologies must be utilized. Fortunately, much of this problem can be solved through database modernization.

Understanding Technical Debt

To understand technical debt, readers should turn the clock back and consider how most commercial applications originating in the 1980's and early 1990's were developed. This was before DB2 for i (as it is known today) or relational database engines reached maturity and before the introduction of the ILE programming model.

Although users of the platform had an integrated relational database engine on the System/38 since announcement (1978/9), due to the immaturity of relational databases, it was initially effectively a collection of "flat files". All entity relationships and data validation rules (valid values, casting, etc.) was enforced within our HLL (High Level Language – usually COBOL or RPG III) code.



Technical debt – why?

Should readers consider that the average LOB (Line of Business) application from this era consist out of approximately 1 500 – 2 500 individual program objects, and then consider that EVERY single program that manipulates ANY of the entities (aka physical files or tables), MUST (theoretically) have EXACTLY the same validations rules and relationships enforced, you can begin to understand the significance of ANY maintenance request that involved a change to a field (column) or file (table). To put this in context – MOST maintenance requests will USUALLY entail such changes…

All readers need to acknowledge that developers essentially coded relational database logic, using our HLL's of the time, due to the relative immaturity of relational database engines at that time. The 1980's and 1990's was a time of dramatic strategic advances in software engineering globally in particular.

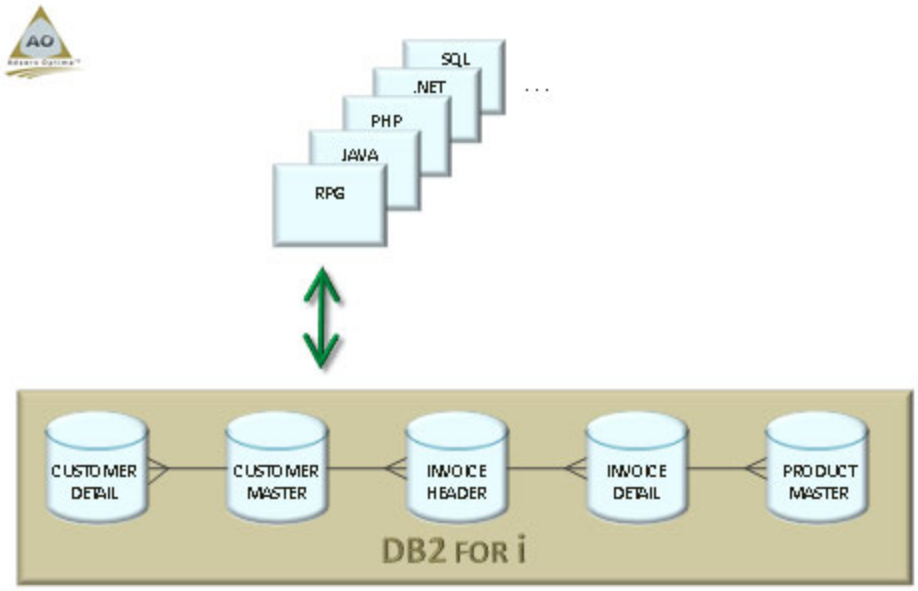
With the introduction of the ILE programming (OS/400 V2R3) model, shortly followed by the new ILE RPG compiler on the AS/400 in 1993/4 time-frame, all the heritage applications of the time was essentially "broken" from an architectural perspective. One of the fundamental considerations of the ILE programming model, was to facilitate "code re-use", having a single instance of a piece of logic, and re-using this logic, when required. It also allows for mixing languages dynamically and sharing job control and program storage interchangeably…

DB2 also advanced dramatically in this same time-frame. This was the "golden days" of the AS/400 (our proud heritage), with rapid development and achievements (CISC to RISC, etc. etc.), especially in terms of sales and excitement around the platform. The remarkable fact (which these days "haunts" us to a certain degree) was how well IBM protected and insulated us from all the changes… our heritage applications happily kept on running, producing the goods, creating the FALSE sense of security that all was well…

Also during this time, our applications moved from essentially online capturing of information with batch processing, to integrated OLTP. It was remarkable how well our applications (and

developers) adapted. IBM Rochester achieved some remarkable feats, allowing the developers and users of the AS/400 and successor platforms to achieve something other manufacturers still dream off… However, our applications were ANTIQUATED in their construction methods and were a ticking time bomb…

Add to this the industry move to client/server in the late 1990's and then the exposure of our applications to the Internet and mobile devices, using fat or thin clients (or mixture), and users had simply too many demands on our stretched development resources. Many changes became very difficult, especially if it involved field (column) and/or file (table) changes, as developers had to find every single instance where that entity (file) or data element (field) was manipulated. If developers missed ANY, data corruption potentially occurred…



Data-centric implementation

Adding to this highly toxic recipe, many of the early adopters of JAVA and other development environments and languages, accessed and manipulated our DB2 files (tables) directly, DUPLICATING the same validation and relationship code (that already existed in RPG and COBOL) – a recipe for disaster and duplicate/triplicate/quadruple/"to infinite and beyond" maintenance burden. Even worse, the opportunity for data corruption was increased by orders of magnitude… And attempting to isolate the "rogue" code inserting invalid data, extremely difficult to find...

If it wasn't for the protection and reliability of this amazing platform and architecture, most companies and developers would have ended up involved in untold horrors…

What should have happened in the mid 1990's, was the gradual modernization of our applications, leveraging the ILE programming model and the advances in the database engine, implementing "data centricity".

Contrary to popular belief, we actually did have implementations of "data centricity" from the mid-1980's, with application development tools such as Genesis V and Synon 2E. Synon 2E (now

CA2E) and Simon Williams specifically was decades ahead of its time, from being completely data centric in its design and coding paradigm.

However, the squeeze was already on in IT spend, with the C-suite expecting more for less… and our applications was happily producing the goods, so why worry???

As a result, very little AUTHENTIC modernization happened, at best a little lipstick (on a VERY ugly pig by now), if the users complained too much about our "green screens"…

As indicated in the previous picture, developers should have started to adopt and implement "data centricity" in the mid 1990's, allowing DB2 to look after ALL data validations and entity relationships, making the database "self-aware" and placing DB2 (and NOT our HLL code) in control of integrity.

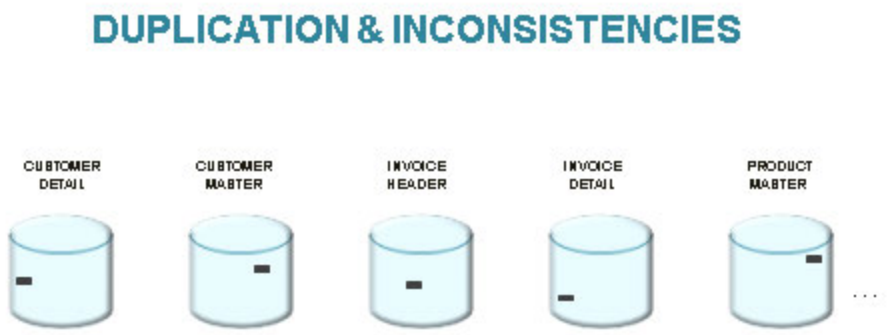
Please note that "data centricity" does NOT mean SQL. And SQL does NOT mean "data centricity". It means that the database is in CONTROL of enforcing all entity relationships and data validations, preventing ANY client (RPG, JAVA, .NET, SQL, etc.) from injecting ANY inaccurate or incomplete data into your database. Data centricity CAN be achieved with your existing applications, improving your data quality and integrity by orders of magnitude.

ALSO, please acknowledge that developers now have SIX (6) "logical" database constructs at our disposal – 3 defined in DDS and three in DDL (DDS View, DDS Join, DDS Multi-Format, DDL EV Index, DDL BR Index, DDL View). The DDL constructs DO NOT replace the DDS constructs. ALL of these constructs are different, each with its unique characteristics.

Additionally, it will facilitate an exceptionally AGILE development environment, allowing developers to RAPIDLY respond to changes in the business environment. An added benefit is that you can use tried, tested and very stable code for enforcing single instances of these rules INSIDE DB2, no longer in our HLL code.

And no matter which "client" (RPG, COBOL, SQL, C, C++, JAVA, PHP, Python, choose your preferred language) access the "self-aware" database and receive the same record or result set. It is ALL about single instance, re-usable code. A single place where maintenance is performed, as opposed to n-times in every program/function accessing the data…

It also implies that the database will GRADUALLY become a true relational database, with proper normalization enforced, long file and field names exposed, etc.



Technical debt – Data quality

An issue few people acknowledge is what has happened to our data quality and integrity, due to DECADES of neglect. Again, due to old programming constructs and development methodologies which originated in the 1980's, it is not unusual to see MULTIPLE versions of the truth of the same data element in our business processes, depending where users find themselves in the business processes. As a result, executives may often look at the incorrect data element when making decisions.

Again this amazing platform SHIELDED users from potential disaster. What should have happened when DB2 on i became the amazing engine it is today, was that developers should have embarked upon the implementation of proper data management and data engineering principles. Developers should have started managing underlying metadata (the data describing our data) and the quality of data a LOT more stringently, as is expected on ANY of the competing platforms.

To allow readers to grasp the significance of what this document is trying to highlight: it is not unusual to see more than 50 000 discrete data elements in "use" in the average IBM i LOB application, whilst experience suggests that 1,500 discrete metadata elements would be the number of metadata elements that one could expect to find in the average LOB application. Developers/application owners can easily validate this, by running the FREE analysis provided by AO Inspector.

Consider for a moment that your developers are using HLL code to keep the contents of these metadata elements in sync, and readers will acknowledge the IMMENSE waste of time that is taking place. Consider then the impact (RISK) of not keeping these elements in sync, or even worse, inconsistencies. Imagine what the quality of decision making is. Extrapolate this with the advent of predictive analytics and cognitive computing. A potential recipe for a very large mess.

It is therefore CRUCIAL that readers introduce proper Strategic Data Management/Master Data Management principles as a matter of priority, or installations and users WILL drown in data,

especially with BIG DATA and cognitive computing approaching installations at breakneck speed…

Understanding Database Modernization

Companies must think strategically to correct the fundamental challenge brought on by lack of application agility, which is retained by old code and an inflexible database structure.

Often, companies start modernizing the UI because it's the most visible part of the application.

A pig with lipstick remains a pig.

The most important piece in modernizing IBM i application assets should be in modernizing the database and eliminating monolithic code so that discrete functions of core applications can be leveraged for a variety of benefits.

In addition, our old application architectural constructs and monolithic code cause unmanageable maintenance burden, which delays change and enhancement requests, which in turn frustrates users and business managers.

In order to create application agility, applications that were designed on old standards have to be fundamentally re-constructed in order to unlock their massive value, retain the competitive advantage facilitated by those applications and recover the business rules in re-usable components.

All modernization has to start at the database engine layer, as most of the advances we have seen introduced to the platform since 2000, pre-supposes the use of the SQL database engine. In fact, in our experience, if the SQL engine is leveraged, your code base can be reduced by approximately 80%, by simply moving database functions we historically coded in RPG (or other HLL), down into the database engine (data-centric implementation).

It is possible to implement a solution with minimum disruption using a low-risk, gradual, iterative process to ensure no loss of business continuity and to support parallel databases during the process. Subsequent refactoring or re-engineering of the code, during a "modernize as you maintain strategy" at both a code and database level will require recompilation of the objects.

Signs that you need to modernize your application database

Operational constraints –This will usually present itself with frustration by your users, or a demand to support a new UI device, a new service, integration to external parties, in the nick of time. Agility and responsiveness to competitive pressures is the key identifier here. Most companies who replaced their IBM i based applications (usually migrating away from IBM altogether), insisted that lack of agility was the primary cause.

Application maintenance is an excessive burden – The maintenance burden is often the cause of people moving off of the IBM i system. Most surprising though is that few of the installations have made the connection between the maintenance burden and the cause of this problem: their monolithic code.

Maintenance backlogs and frustrated users – IT developers can simply no longer cope with the maintenance burden of their system, as they not only have to effect the change, but they then have to test every single piece of code that was changed. What usually happens is that the end-

users of these systems become frustrated, as changes simply take too long to implement in the very demanding business environment of today.

Cannot find quality resources to support it – Not everyone can easily maintain your heritage IBM i applications accurately, understand it completely and have the programming skills to concisely adapt the application, due to significant hidden technical debt. These rare resources are scarce and can be costly.

No executive buy-in – Most people are afraid of what they don't understand. With a modernized system, new executives are more receptive to IBM i technology and all of the benefits that you know so well.

Data integrity issues – New developers over the years (and years of neglect) may have affected the integrity of your database. In fact, it is highly likely that duplication and inconsistencies may have occurred over time at a metadata and structural metadata level which has compromised the integrity of your **database**.

Lack of agility – Your business may be constrained, held back by the lack of flexibility within your system. If you can no longer respond in an agile fashion to changes in the business environment, that's a huge problem. If companies cannot respond rapidly to changes in the business environment, companies are often doomed. As a result, executives and users start looking for alternative application solutions that facilitate rapid response to changes. Lack of agility also includes being held back from technology advancements, by the requirement for a modern graphical user interface, for 24x7 access via mobile devices, smart phones and web browsers.

Difficulty in users extracting their own information – As the file (table) names and field (column) names in heritage applications are defined in short, technical abbreviations, users have immense difficulty in associating with technical terminology. This is complicated even more, with "several versions of the truth", due to the duplication of fields that exist in multiple tables, due to historical design and coding practices. Added to this, little to no database normalization occurred, with even less provision for proper indexing (logical) constructs, to allow DB2 to efficiently retrieve the correct datasets. These factors makes it problematic for users to easily extract their own information in the format they require it.

Constrained implementation of native analytics and cognitive computing capabilities – Due to poor design and normalization, as a result of decades of neglect, introduction of the newer DB2 for i features, such as OLAP, is problematic. The underlying foundation (the database) of the heritage applications are so burdened by technical and integration debt, that sensible extraction of valid information from the application "databases", is exceedingly difficult.

Workload consolidation – Clients running multiple servers want to increase the utilization of server hardware by allowing one physical server to host multiple virtual machines. This allows the server to operate its original workload as a virtual machine and host additional virtual workloads simultaneously. At the same time, clients want to consolidate legacy databases and applications. Furthermore, power and cooling demands and maintenance costs are lowered, translating to lower operating costs and lower capital-intensive projects.

No Central Metadata Repository (Data Dictionary or FRF) – Due to the extended period that IBM i applications have been in production (it is not unusual for IBM i applications to be 25+ years in production), with significant contribution of staff turnover during this time, it is seldom that development staff and IT management have insight into how their database have devolved. They seldom, if ever, have a single place where ALL the metadata elements are available and under

management control. Even worse, little formal Strategic Data Management (SDM) or Master Data Management (MDM) management disciplines are in place, as the platform has been so forgiving. In the modern, data-centric world and with the associated impact that BIG Data will have on all commercial applications, a central management platform is critical. If you don't implement this, you will soon lose control of your data.

The Value of our Data and Infonomics

It is CRITICAL for readers to acknowledge that business data is the lifeblood of any organisation and that especially the C-Suite essentially hold the business data (hence the future of the organisation) in the palm of their hand. Ignore this at your peril.

Only imagine for a moment what will happen to your business, if you had to lose all your application data and information in an instant.

The VALUE of your data to your business will be the best possible motivation to decide whether or not it is worth investing any effort into it, to improve it and the access to it.

The subject of Infonomics is a massive and rapidly evolving one, as the principles and formulae matures. Our best recommendation to position this correctly, is to suggest you read the following exceptional article on the subject: **"How Do You Value Information? By Alex Woodie, interviewing Doug Laney".** A link is also available on the AO website.

Additionally, the publications by the following thought leaders may be able to assist you:
Martin Doyle – DQ Global
https://www.linkedin.com/in/martindoyle

Doug Laney – Gartner
https://www.linkedin.com/in/douglaney

Bernard Marr – Advanced Performance Institute
https://www.linkedin.com/in/bernardmarr

Jay Zaidi – AlyData
https://www.linkedin.com/in/javedzaidi

The VALUE of your data and the quality of your data may likely be your best possible motivation to do something about modernization of your heritage applications.

IBM i Strategy and Roadmap

IBM has introduced a staggering amount of major database technology enhancements since 2000 (starting in 1995/6) and over 95% of announcements on the topics of IBM programming languages, programming model, operating system, database engine and application development since 2000 presupposes the use of the DB2 SQL (SQE) database engine personality on the platform.

Despite IBM's indication that SQL is the strategic database interface for IBM i, the bulk of all customers still use record level (RLA) or native IO access (F-Spec in RPG) as the primary database access method and DDS as the data definition language, with tight, compile-time integration. This means that although the installed base has one of the most advanced implementation of the DB2 database engine at their disposal, they are using the DB2 database engine severely "shackled" or constrained, because it kept on processing transactions the "old" way.

Legacy database engine access (CQE and RLA)

Prior to the advent of the latest generation of relational database engines and modern programming techniques, application programmers were forced to physically code all the relational logic (i.e. the relationship between customers, all valid customer delivery addresses, all orders for customers, all products per order, etc.) as well as all the database validations manually. This is in addition to the business-unique application logic (for instance how orders are fulfilled, from which warehouse products are shipped, how picking is optimized, etc.), which usually translated into the competitive advantage for that particular user of the application.

As a rule of thumb, about 80% of the lines of code of any "legacy" commercial application, is focused on entity (database or "object") relationships (such as customer to customer locations, customers to open orders, order lines to orders headers, products to orders, etc.) and validation rules (product number to a valid product, values chosen within a valid range, Zip code - a valid Zip code, Social Security number - a valid Social Security number, etc.).

As these relationships and validations have to be maintained in every single program that manipulates these files, it can become a very tedious and laborious process to update these efficiently. This is where a lack of application agility can bog a system down as every instance of a validation or relationship has to be identified in the entire application in order to effect a change.

Separation and de-coupling of the database from our application logic

A fundamental requirement of any authentic modernization process, demands the implementation of multi-tier architecture (also referred to as "MVC", for Model-View-Controller) and especially the "de-coupling" of the database from the application logic. The driver behind this, is the requirement to respond rapidly and in an agile fashion, to changes to the database.

This is a crucial requirement, as the database must function as a complete, "self-aware", and "self-enforcing" resource.

Why SQL?

Regardless of IBM recommending SQL since 2000, one question arises: Why move your application's database objects from the old DDS definitions to native DB2 DDL definitions (SQL)? IBM said it themselves in the whitepaper 'DDS and SQL – A Winning Combination for DB2 for i':

"Those IBM i clients who have failed to keep track of this SQL metamorphosis are missing out on a plethora of new SQL-based functions that can make it easier for their IT teams to meet the ever-changing list of business requirements."
(A copy of this Whitepaper is available for download on the AO Website).

IBM recommends data-centric programming in which you allow DB2 to perform base data processing using the immense capabilities of DB2 for i. The benefits of using SQL include:

1. Improved agility - Flexibility and openness through SQL allows companies to be agile and regain their competitive advantage.
2. Utilizing the latest technology advancements from IBM - The DB2 for i SQL personality has been the foundation of all developments and enhancements to IBM i (and predecessors) since 2000.
3. Ability to present a modern database, improve data integrity and enhanced analytics.

4. A plethora of documentation, database tools and resources are available due to the widespread, platform independent knowledge of SQL. This provides access to a massive pool of younger generations of developers, arresting the risk of an aging developer demographics, with a large percentage of developers fast approaching retirement.
5. Significant performance enhancements. Using SQL can significantly reduce the size of code, which improves performance and reduces maintenance backlogs dramatically.
6. The integrated data-centric components and analytics (both OLTP and OLAP) functionality provides significant and exceptionally powerful native (read high performing) analytical processing capabilities. This facilitates the introduction of cognitive approaches in heritage applications.

Should the staggering list of DB2 for i enhancements since the introduction of the “COBRA” DB2 (IBM i V5R4M0) database engine in IBM i be studied (separate document available upon request), ignoring and not using DB2 for i properly, borders on criminal behavior in our opinion. It also stifles your business substantially.

Benefits of database modernization

Modernizing your database allows you to preserve your investment in your IBM i applications and retain access to all of your heritage applications and data. That alone provides significant savings, but there are many other benefits of database modernization including:

Agility	- Rapidly respond to changing business needs - Build and efficiently maintain new applications
Integrate	- Integrate data from core, legacy applications with other applications and functions that reside on other platforms
Mobilize	- Mobilize your applications across all mobile and traditional devices anywhere, anytime
Cloud	- Greater application scalability, maintainability, upgradability from cloud based applications
Performance	- Deliver huge improvements in end user application performance
Analytics	- Leverage analytics to predict and solve complex business problems, improve decision-making and realize cost savings.
IBM PureSystems	- Integrate with other enterprise applications and IBM PureSystems ready

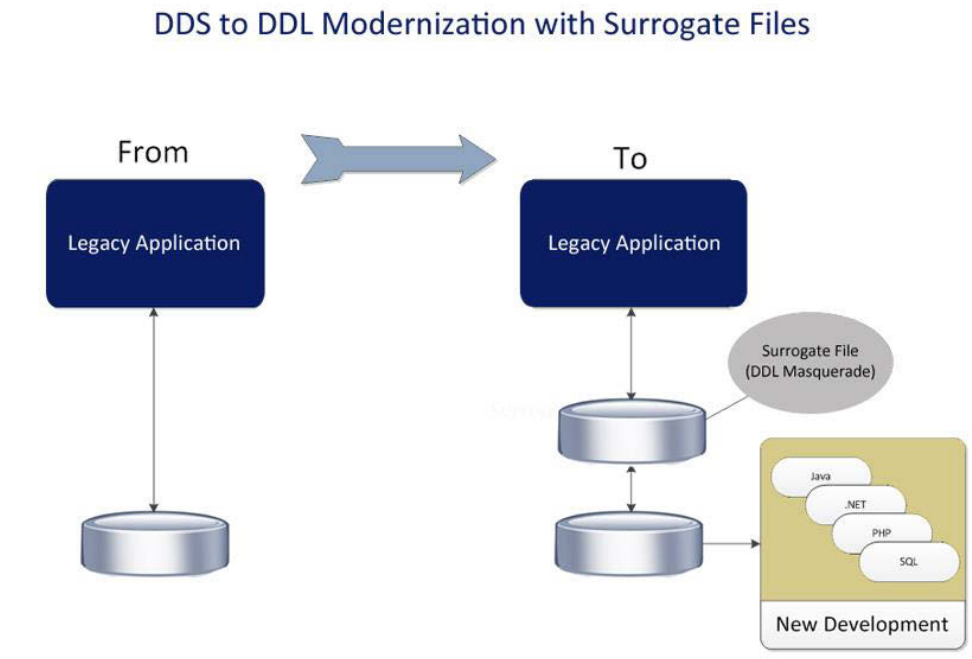
Methods for Implementing Database Modernization

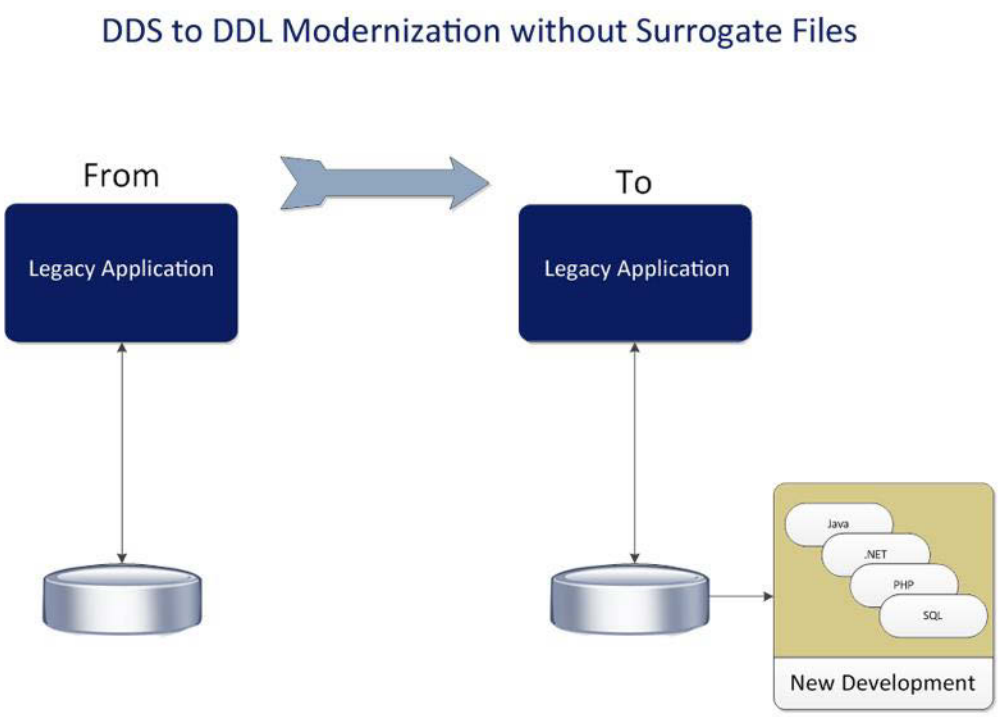
Different methods exist for modernizing the DB2 database from old style database definitions (DDS) to SQL (DDL). It is important to understand these different technology options and know the pros and cons of each. Fortunately, new technology has been introduced that changes the process, making it a low risk, gradual, non-disruptive and relatively simple process. Before we look at this new technology, let's look at the other options which have been known to introduce challenges and inefficiencies.

Use of Surrogate files

The use of surrogate files has been recommended in the past by various software vendors as an alternative to the native conversion of the underlying database infrastructure. "Surrogate logical files" masquerade as physical files to heritage applications thereby allowing the heritage applications to access the "new" database files without the need for recompilation.

While the use of surrogates are beneficial when new applications are being implemented and heritage applications remain unchanged ("frozen" in time), bear in mind that approximately 80% of the lines of code (all lines of code implementing validations and enforcing data relationships) currently in your heritage application, will eventually and over time end up in the database engine (should you accept IBM and our recommendation to implement data-centricity). Then consider where you want to go with your heritage application ("freeze" it) versus leveraging the immense value both in data/information, as well as your competitive advantage encapsulated in your heritage application. Implementing a native migration will provide a natural long-term strategic foundation to gradually remove all operational constraints inhibiting your agile response to changes in the business environment. Also decide if you want the surrogate approach which limits heritage applications or if you want the full might of the DB2 athlete at its disposal.





Other Options and Challenges

Access to the source code – Some vendors require access to your source code for modernization implementations. It's important to remember that changes to your source code may force you to be locked into a particular vendor.

Disruption of users – In the past, with a database modernization implementation, the disruption to users and the business were significant. In addition to running two systems in parallel, companies were forced to pay double maintenance with any new or interim solution.

Old programming tools and anomalies built into the system over time – Due to the sheer age of the applications and regardless of the management practices, a LOT of anomalies have been introduced into our systems over time. Some of this was caused by "emergency" maintenance, but a LOT of it is simply caused by neglect – inconsistent validation and relationship rules in our code.

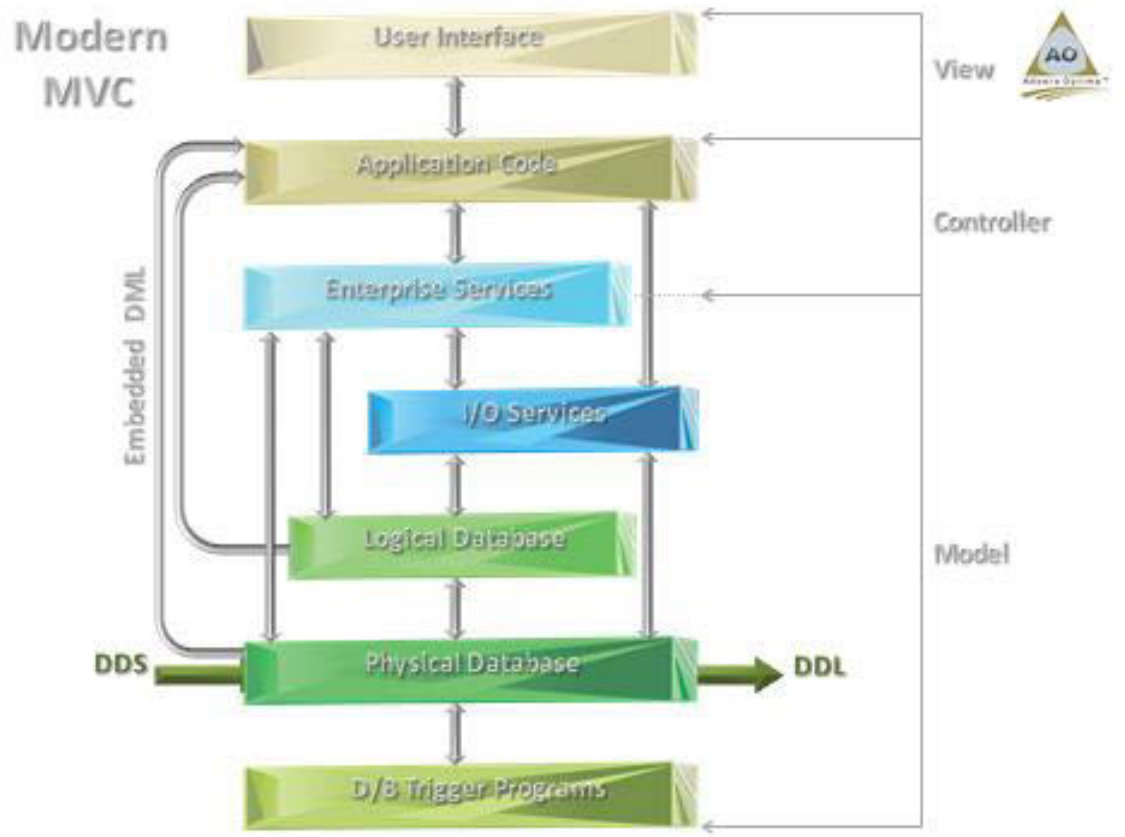
Native Modernization Implementation

It is our belief, confirmed by extensive experience and remarkable success, that the only lasting modernization strategy has to start at the fundamental database definition level, and especially with your underlying metadata. Any other adjustments or adjustments are tactical moves at best. They will not in the long term remove the fundamental barriers to a permanent solution. The challenge is how to achieve this with the minimum disruption to your users. Native modernization provides the lowest risk for a solid, long term foundation, from which you can then start to extract maximum benefit out of your heritage application. It is imperative that you transfer your database definitions into DDL, with the bulk of your relationships and validations moved out of your application logic into the DB2 database. This will provide you with the foundation to start leveraging the incredible capabilities of SQL and the DB2 for i database engine. DB2 should be

allowed to do all the “dirty” work (or “heavy” lifting) for you, so you are free to focus on delivering innovative business solutions and logic.

Multi-Tier architecture - Multi-tier architecture (or MVC in technical parlance) in modernized applications allow companies to achieve separation of code in the database (M), user interface (V) and business logic (C) that result in more agile applications and improved maintainability.

This illustration shows what our optimum application architecture should look like and which elements are non-negotiable going forward. By implementing this architecture, we can reclaim our rich heritage, whilst removing those building blocks that have caused a LOT of pain for IBM i developers. It is significant that if modernized properly, you may end up maintaining as little as 10% of the lines of code you had to maintain originally in your heritage application.



No vendor lock-in, complete control – We believe that you should never exchange one dependency for another - quite often more proprietary than the previous. Take ownership of your applications and the destiny of your systems. With native database modernization you retain complete control of your database. You receive full management capabilities of the entire database, controlling who may change structural metadata, providing a full audit trail, etc.

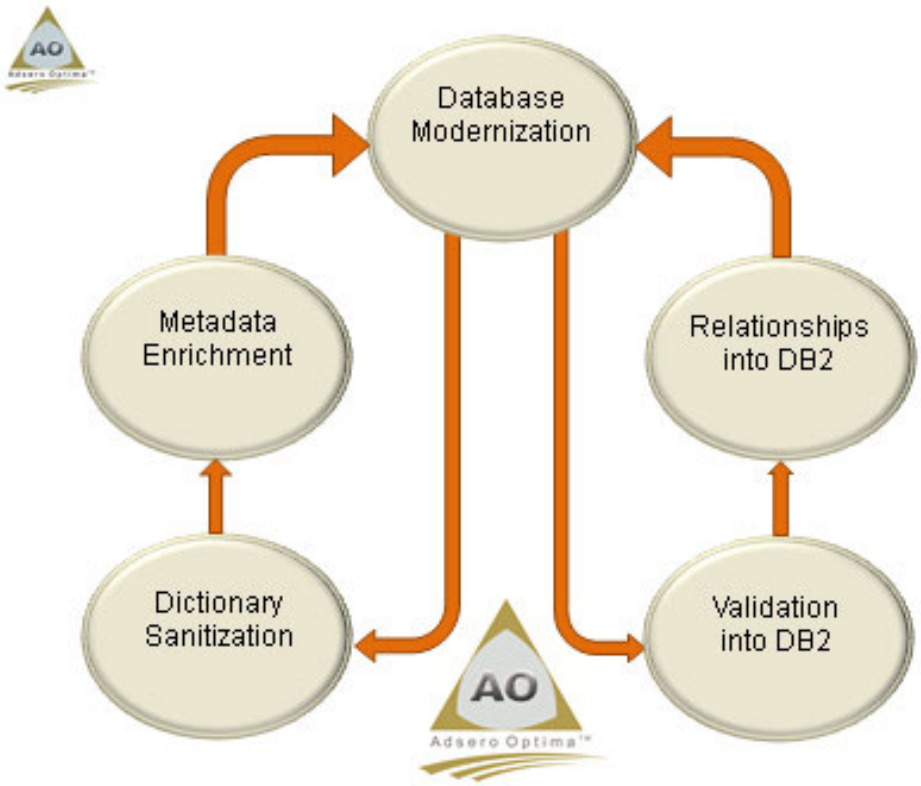
Non-disruptive, low risk – Designed to facilitate a completely transparent, non-disruptive migration from DDS to the native SQL engine offers a low risk implementation with little to no downtime for users.

No LVLID changes in Phase 1 of database modernization – No LVLID changes during Phase 1 (initial DDS to DDL migration) of the database upgrade process is crucial, hence no recompilation of any code. This facilitates a low risk, easy and rapid diagnostics, as only ONE variable changed – the database.

Be very, very cautious of mass recompilations, suggested by some. Due to the age of our heritage applications, and the relatively low adoption of stringent CCM (software change and configuration management) tools or processes in many of the IBM i development shops, the source code you consider recompiling, may (often) not represent the actual object currently in production. We have seen too many occurrences where the deemed source and objects were not the same.

It is fundamental to limit the change to one component (the database) only in the initial upgrade, to facilitate easy problem determination, in the event of any migration/upgrade failures.

Gradual sanitizing of metadata, structural metadata and database – One of the key considerations of our entire modernization philosophy and roadmap, is to gradually, modernize your system as you perform routine maintenance. The benefit is immense and you implement the very desirable "continuous improvement" philosophy.



Small incremental steps, fast ROI – We believe that you should introduce a long term strategic roadmap, focusing constantly on ROI and addressing the fundamental causes and perceptions of your system being regarded as "legacy". We always approach this from the 80/20 rule perspective, where experience indicates that 80% of your transactions are generated by 20% of your application. These programs and associated databases can be migrated quickly and with

significant ROI, as foundation to start familiarizing your developers with data-centric development paradigms, but must form part of a long-term strategy.

Central Metadata Repository (Data Dictionary) – Fundamental to your database, the data quality and data integrity considerations are of utmost importance, as it influences most business decisions. Due to decades of neglect and the platform being so forgiving, your developers should immediately embark on the implementation of a central Data Dictionary. This should then be used to identify all the duplication and conflict that exist within your heritage application databases, to start a concerted data quality and data consolidation improvement project.

AO provides a FREE tool (AO Inspector), which can be used to analyze your database and the underlying metadata, providing you with exceptional insight and statistics of what your database really looks like.

The central metadata repository become the foundation and source of all your data management objectives, business rules, attributes, usage, etc. – in short, the key to your entire application.

Implementation of a "Data-Centric" Design and Development Approach – The value of data-centricity and model based application development is widely recognized. It facilitates an exceptionally agile development environment, with remarkable data quality and data consistency benefits.

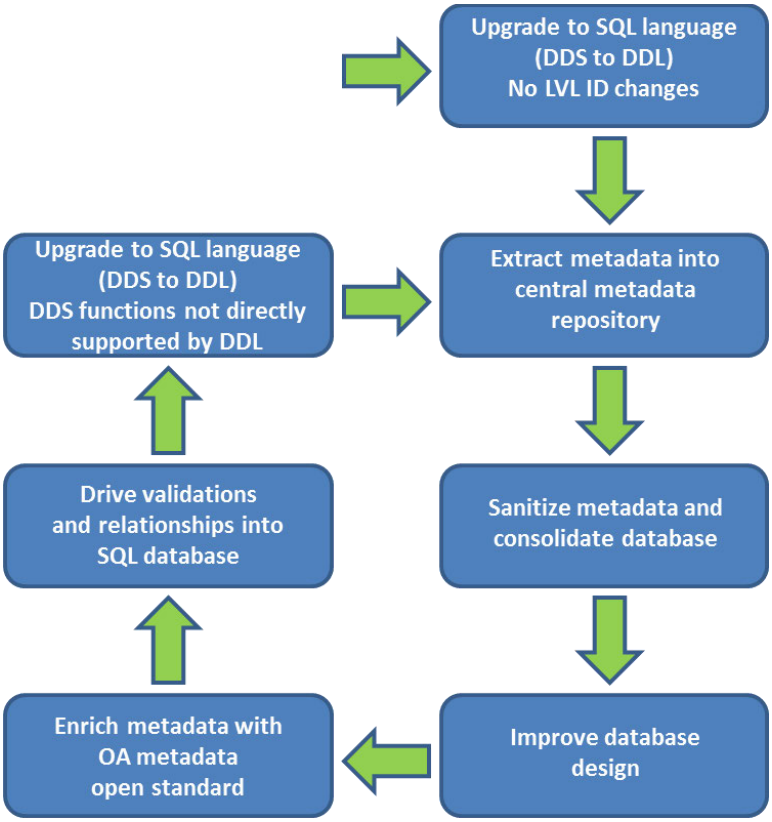
Modernize as you maintain – Whilst gradually moving to a data-centric development paradigm, with the objective of allowing your development staff to familiarize themselves with the latest software engineering approaches, provides a natural, low risk adoption roadmap. A major benefit of this approach is that all modernization efforts are immediately focused on those components of your application that inhibit or frustrate your users.

Be extremely wary of "analysis paralysis" to set in or "Big Bang" projects. Both are high risk, as the size and age of our applications cause most developers to tremble in trepidation of the consequences. Break it down into small consumable projects, aimed to alleviate immediate business demands. The benefits and confidence this develop for your developers and users, will exceed your expectations.

For more information, please see the "The standard AO Modernization Roadmap customized to individual installation requirements for maximum ROI" document on the AO Website for more detail.

Single, Central Management of ALL Database components (Metadata, Tables, Triggers, Constraints, IO Services, all logical constructs and Enterprise Servers) – An extremely important consideration in the data-centric development paradigm, is a single management interface of all the components that defines your database and applications. Additionally you need to ensure and verify the integrity and completeness of all components, during the day to day operations and usage of your applications.

How to migrate to SQL

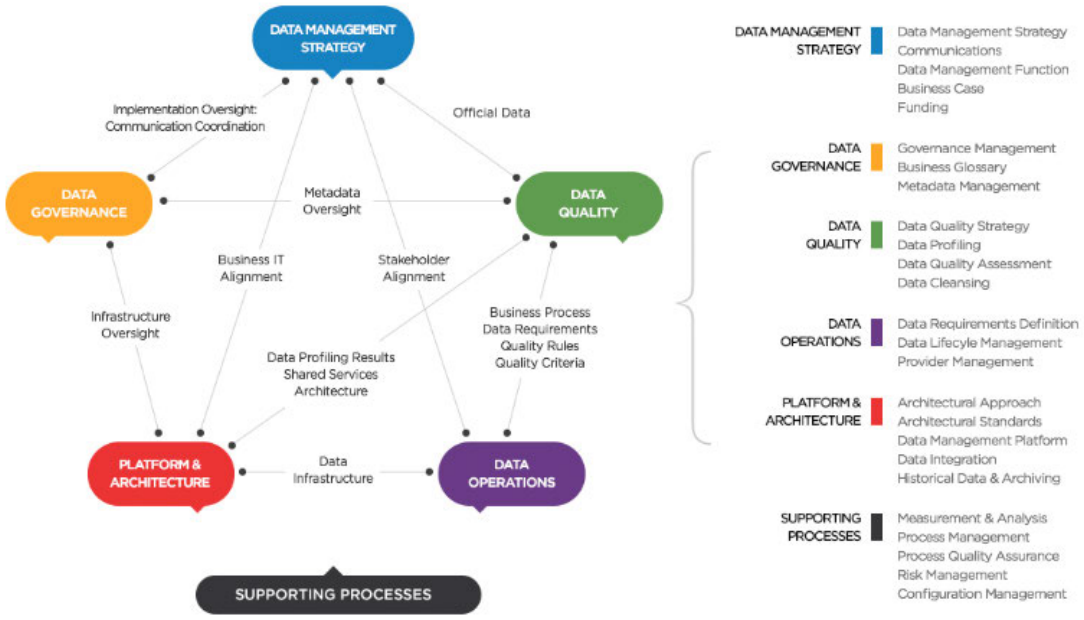


Additionally, there is no need to modernize the entire system. In our experience, the most sensible way is to consistently consider the ROI and benefit of modernizing a specific function. The perfect initial candidates are the 20% of business function that generate 80% of transactions. Modernizing those first, will change perceptions and reduce your maintenance burden dramatically. Other candidates for modernization are the programs that require continuous maintenance. You can track these down by carefully analyzing your maintenance requests. By refactoring these programs, operational and architectural deficiencies can be corrected.

Clearly so-called CRUD (Create, Read, Update, Delete) or BREAD (Browse, Read, Edit, Add, Delete) functions which provide relatively little in terms of ROI and should be left for last, or whenever you have a maintenance request.

Integrated Data Management Practices

Due to the strategic importance of your business data, which is to increase in importance by orders of magnitude due to the introduction of big data and cognitive computing principles, formal data management principles and resourcing must receive significant C-Suite consideration.



Source: [Jay Zaidi]
[AlyData]
[http://cmmiinstitute.com/data-management-maturity]

We concur with IBM Rochester on the importance of the DBE (Data Base Engineer) function, which does not necessarily require new headcount. We believe most development installations already have someone who functions as the natural “go-to” person around any subjects around the database. The importance of this function should not be underestimated, It is critical to the strategic success of your business, especially in the rapidly changing business environment.

What are some of the benefits of adopting the AO approach?

1. Native DDS to DDL upgrade/migration
2. Central Data Dictionary
3. GRADUAL Sanitization of Metadata and structural Metadata
4. GRADUAL automation and exposure of long file- and field names (table and column names)
5. GRADUAL consolidation of database model
6. GRADUAL metadata enrichment as per OA Metadata Open Standard
7. Automatic shadowing of n-instances of same Schema
8. Central management of ALL database components – Tables, Constraints, Triggers, IO Servers, Logical components (EVI, IDX, Views {DDS & DDL), DDS Joins, DDS MF), Enterprise Servers
9. Continuous verification and housekeeping of databases in production compared to structural metadata
10. Central repository of all metadata components and where used
11. Central database workbench – ongoing management and maintenance – new and old
12. A completely modernized application at 20% of the cost of replacement or redevelopment

The following video may also provide you with additional background:
https://vimeo.com/93652272

Conclusion

IBM i is a very powerful platform. Despite the fact that many advancements have been introduced over the years few developers have appreciated the true value of these advancements and continue with systems that are slow and difficult to manage - perceived as “legacy” primarily due to a lack of application agility.

Organizations migrating legacy database objects from the current format to native DB2 SQL have experienced many advantages. They are able to benefit from the significant value invested in their heritage applications and at the same time eliminate unwanted constraints.

By implementing native database modernization they are able to extend the life of their heritage applications into the future while remaining competitive and take advantage of many new technology advancements such as Business Analytics, Mobility, Cloud Computing and PureSystems. All of which can now be leveraged because database modernization has taken place.

Through native database modernization, it is possible to implement a solution with minimum disruption using a low-risk, gradual, iterative modernization process to ensure no loss of business continuity and to support parallel databases during the process. This automatic, gradual and non-disruptive process allows implementation without the use of surrogates, without Level ID changes or access to source code. This can be achieved without any risk or down time.

The final result is a modern application architecture with a clear separation between the database, the user interface and business unique application logic. This means that companies can reclaim their agility, regain their competitive advantage and respond more rapidly to changes in the business environment.

One question remains: Are you going to gain a long-term competitive edge with your IBM i heritage applications or lose the significant value that’s still left?

Sources:

1. Gartner Executive Programs' Worldwide Survey of More Than 2,300 CIOs Shows Flat IT Budgets in 2012, but IT Organizations Must Deliver on Multiple Priorities http://www.gartner.com/it/page.jsp?id=1897514
2. DDS and SQL – A Winning Combination for DB2 for i
3. For these and an extensive list of other reference works, please visit the Adsero Optima website at www.adsero-optima.com, as the list is extensive and evolving, especially from a data quality, data integrity, data-centricity and a SDM/MDM perspective.

About TEMBO Technology Lab (Pty) Ltd

TEMBO Technology Lab (Pty) Ltd specializes in the development of database modernization and management solutions for IBM Power Systems running IBM i. Our flagship product, Adsero Optima™ (AO) (www.adsero-optima.com) is considered unique in its field globally. Adsero Optima modernizes legacy databases to the powerful SQL (SQE) engine whilst facilitating the implementation of a "data-centric" architecture, preserving and extending legacy applications and providing a solid foundation for future modernization technologies and projects.

Our primary focus centers on full spectrum modernization deployments that result in improved IT efficiencies, specifically around DB2 for i SQL migration. We offer deep expertise in Design Recovery, BPI, Business Agility, SOA, SaaS and Cloud Computing. To fully exploit the power of IBM i, we leverage data-centric ILE-RPG IV design paradigms, embedded SQL to some extend where it provides flexibility, SQL (DDL, DML), Stored Procedures when needed, and NATIVE, IBM i based responsive HTML interfaces.

We have experience with most UI tools (screen scrapers), Zend framework, PHP, JAVA, AJAX and RPG Open Access (ROA). We are dedicated to providing world-class technologies and providing the skills and experience needed to unlock the true value of software systems on the IBM i platform.

As a company, our RPG and DB2 skills are recognized internationally. For more detail, please visit http://www.tembotechlab.com

Allow us to demonstrate the value of this network to you.

Reclaim Your Heritage. Unleash Your Data Value.

AO-TEMBO iModernize : https://www.linkedin.com/groups/1231677

V4