Table of Contents

Revision History

Version	Date	Author(s)	Revision Notes
1.0	6/30/2017	Kelly Nolan	
2.2	8/1/17	David Pepper	Prepared for 8/2 meeting
3.0	8/7/2017	David Pepper	

Project Overview

History of DMV Access 97 Applications

Microsoft Access is a personal computer database application that can scale from a single-file pattern to a split design to serve small workgroups over a network. Since it was invented before Active Directory overtook Novell in enterprise login, it had a file-based independent workgroup authority to secure group installs until the feature was finally written out of the product in 2007. Beyond the fileserver, there is no real server -- all processing happens on the client, including managing record locking and authentication, with all interactions trigged by events on the client.

Over time, best practices in Access development have evolved to overcome various shortcomings of the system by allowing many original functions to be integrated with other products, while keeping forms and compiled VBA modules local and working as always. Split database designs can store the back end data in SQL Server. Domain authentication allows single sign-on for users and transfers administration of security group membership to Active Directory. Finally, many advantages come from the modern database server. Among these are that business logic can be stored and served from there to multiple applications including other platforms, performance of server-side query processing is *way* better than client-side, and client applications have fewer roles to play and can focus on transactional database application interactions. All kinds of reporting process and data interchange are more elegant, secure, and fast on the server too.

As of August 1st, 2018, The State of Vermont Department of Motor Vehicles (DMV) has a collection of approximately 29 enterprise applications in active use by ~260 users at 11 sites, which were developed in Access 97, as well as approximately 10 obsolete Access 97 applications with data that needs to be retained. This portfolio of DMV “split design” apps uses Access workgroup security, and have seen very little creative attention – they are in “maintenance mode”. As a collection, they also eschew any pattern of categorization, serving many workgroups, ranging in size and complexity, some tracking money, some are systems of record, and some not. Basically, the unifying feature of the DMV Access 97 portfolio is that it contains apps that didn’t make sense to develop in the mainframe, or later as a web application. This means that many of these apps exist Access only because it was the technology of its day despite now being a better match for another technology; these apps should eventually be migrated to their ideal platform. And some apps are actually well-served by Access’ strengths: easy forms and reporting, user querying, and local storage; these apps should remain in Access.

Microsoft's "Mainstream Support End Date" for Access 97 was January 31st, 2004, and while many companies use Microsoft's business product support lifespan (a minimum of 5 years mainstream, which is sometimes extended for an additional 5 years) as a planning milestone, it is also not unheard of for companies to take strategic risks and to continue to use products well-beyond the support lifespan -- particularly if there is alignment between the development tools and the operating system they target. This is the situation that the DMV evidently found itself in; addressing the dispositions of the portfolio of Access 97 apps eventually became part of the VT Drives project, but were tabled when the project was abandoned in 2012. As extended support for Microsoft Windows XP Service Pack 3 ended April 8th, 2014, like most companies, DMV standardized their desktop platform on Windows 7 Professional (the latest widely popular enterprise OS). Meanwhile the Access 97 applications continued to languish.

The position that the DMV faces in supporting the legacy platform is becoming increasingly untenable, especially with the continuous change of newer and more diverse operating systems. For all these reasons, there is now a fresh consensus that DMV should rapidly resolve the use of Access 97 by moving the current portfolio to Access 2016 -- and that this critical need takes precedence over redesign. Given the effort it will take to migrate and the strategic stake that ADS, AoT and DMV have made in SQL Server, it makes sense to migrate the tables to SQL Server. Since workgroup security has been removed from the Access product there is no alternative but to implement domain authentication. And, since the apps have such diverse purposes, and have in many cases split needlessly into write/read versions, each application should be coalesced back in to a single app client.

The goal above else is to continue service while safely and rapidly removing Access 97 from the State of Vermont’s systems – this is the hard work of the migration and has to be the current focus. However, to the extent that it is possible, each designated app should be migrated and positioned with an eye toward its future destiny and its holistic purpose – including revenue, cost, risk and opportunities. Subsequent projects will cut a path for some apps to go into adjacent platforms (web, Sharepoint, parameterized reports, mobile, etc.), and our emerging strategy will continue to lead us to separate transactional databases from systems for reporting/analysis.

MS Access’ Achilles Heel: Multiple Version Support Problems

Despite many advantages to migrating, there are some substantial forces holding the status quo in place for companies with a legacy Access portfolio. The most significant obstacle is that Microsoft Access (especially older versions, like 97) has earned a reputation for being difficult to support with multiple versions on the same computer. In fact, these problems occur with many Office applications, caused by conflicting application versions and mismatches between files and the applications that create and operate them -- but they also tend to resolve without need for IT support. By comparison, Access is so intolerant of mismatched application and file versions, that the accepted practice is to explicitly hard-code shortcut that opens an app file with the specific executable file (msaccess.exe) of the Access version that is supposed to operate it. This arrangement side steps the computer's registry -- which in the absence of a verbose shortcut is used to decide which application will open a file that an "open event" has been triggered for (typically by a user double-clicking a file).

However, it is difficult (if not impossible) to prevent interactions with the registry on a computers with more than one Access version installed, often setting up a cascade of unpredictable results. Multiply-installed Access versions each claim the registry setting for Access files in succession as they are used. This means that otherwise similar attempts by users to open the same target file can potentially trigger different results depending on which Access version attempts to open it (and there have been 11 versions of Access so far!). When a later version of an Access application attempts to open a file from an earlier version, it offers to convert it -- often after a lengthy delay. Likewise, when an earlier Access application attempts to open a later file version, it will simply refuse to open it.

Meanwhile, group policy has trended toward simply denying users the ability to make changes to the registry. While it is conventional wisdom that it is probably better to deny the registry change than to allow it, group policy owns a big part of the haze of confusing error messages, non-responsive computers, and unplanned restarts.

In short, these problems are very hard to troubleshoot and often have differing symptoms between various computers, users, Access versions, and operating systems. For a counter-based customer service operation like the DMV's these kinds of problems need to be explored critically and strictly kept to a minimum. Luckily, third-party tools have coevolved with Access and are available to address these issues. Total Access Startup (http://www.fmsinc.com/products/startup/) is our last greatest hope for overcoming the multiple version support problems. It promises to keep Access versions from interfering with each other, and automates app deployment as new versions are released. It seems to have a central authority on a file server which, and a user client on every users' desktop.

Active Directory Design for Installation

In the legacy AD deployment design for the Access 97 applications, membership in a single group ("DMV-Database…") grants fileserver permissions, to each application's source folder in a folder on a fileshare, typically mapped to Q:\. Logging into the computer triggers a scripted comparison between the user's local app file (deployed in C:\database) vs. the file-server's application "program" folder. Updated server files are copied over the user's local files as-needed. This means new versions are normally deployed the following morning, but are available sooner to users who log out and then back in.

Membership in an AD group is also used by PC Support to automate the install the Access Runtime software, though it is unclear whether it is membership in the "DMV-Database…" AD group or an administrative group/account such as the "clone" accounts triggers it in practice.

> ***In the course of the Access 97 migration to Access 2016, the design and administration of app deployment will be ceded to DMV Application Development by Infrastructure. The AD design and administration that governs installation of Access will remain with PC Support and Infrastructure.***

Active Directory Design for Security

With the introduction of the .accdb file-type in 2007 (replacing JET with ACE database engines), MDW security was entirely removed, forcing a transition to SQL Server with AD security for companies that needed it. This probably had the effect of discouraging migrations, but once it is in place it comes with many benefits and is considered superior. Among the benefits are single sign-on, logging with non-repudiation, inherited password policy, and improved security. However, it requires AD administrators (or their delegates) to assume an essential role in the day-to-day process of user administration (e.g. database security administration). And, it contributes to the complexity of the AD design, which already has many jobs to do.

In the course of the Access 97 to Access 2016 migration, security administration of users will be ceded to PC Support by Infrastructure. The security design will remain with DMV Application Development.

In the legacy AD design for most of the Access 97 applications, users across the whole ~260-person workgroup typically shared a single MDW account, called "open", secured by a shared password which has never been changed. Effectively this made the "DMV-Database…" AD group for each application the only security for data access and change, and precluded the possibility of achieving non-repudiation through logging. Note: Five of the ~25 migrating apps have some records that in some way further establish a user's identity, such as a "login" table. These outlier app designs have not been evaluated, and may affect the planned AD design for those applications, as they are migrated in turn. Note: the cashier's suite (alone?) uses MDW security with unique user accounts and passwords, though it is not expected to migrate.

The plan is for users to be enrolled in an AD group for their instance, application and level of permissions (limited to just "read" vs. "write" for now). In the rare cases where a legacy user has both the primary application and its complementary "view" application, they will be granted "write" permissions.

Instance Design

The current plan is for applications to be successively migrated (one at a time) through three instances: Development Access 2016, Test Access 2016, and Production Access 2016. In new prod and test there are ~37 security groups planned to organize users of the ~25 apps (all apps have a "writer" group, and some also have a "reader"). Planned membership in prod will eventually be the whole ~260 users, of course. In test, small groups of users will be marched through membership in groups paralleling those in prod, installing the database and software, and allowing testing – the group membership of users' accounts will be active only while users need to evaluate software or its installation. Besides allowing separate groups of user collections, the instance design also organizes and exercises our tasks for data migrations, software installation and uninstallation, and security group un-enrollment and deallocation – all of which need to be done on a tight schedule during downtime at go-live (and ever-after) and which benefit from systematic practice and separation of concerns.

Infrastructure pushed back on the plan to request extending AD groups (by ~37) and AD users (by ~75 at the time) into dev to parallel what was being requested at the time in test and new prod. In current thinking, this would have resulted in the additional ~37 AD groups as well as complementary solitary member test accounts (+~37). Taken to an extreme, this denial ("no test groups/members") would result in a 2-tier development model for deployment, software installation, and security design – and then would presumably carry through to production as some Access 2016 apps would have gone live and begun to need new development and maintenance. In this world, dev is for developers to use -- but peculiarly -- only with their own accounts and groups they are members of (typically with elevated permissions).

The problem is that across AD/SQL Server all permissions are cumulative (except deny) and a user gains its ability to interact with a database by their AD groups being granted permissions in SQL Server. The SQL Server admin also has a strict policy of having "no users" in SQL Server. Meanwhile, our own membership in AD groups let us do everything that makes us us. It is really imperative for developers to have dedicated AD accounts to "test" with in dev with because we are typically logged in as ourselves during development. While there are many functionalities that can be coded/evaluated/operated with elevated permissions, there are many that cannot (e.g.: read, but not write, install runtime version, etc.). Likewise it is imperative to have remotable computers to log into so we can isolate install issues, and establish credentials as users do at signon.

The most likely compromise will be for all SQL databases to grant permissions to a giant roll-up security group in dev for each level: SQL-DMVallappsWriter-Dev, SQL-DMVallappsReader-Dev, and then in turn have a single test account for each group (which could be shared among developers because we never work on the same application at the same time). That would allow us to develop and test installation and deployment in parallel with test and prod. We will still have ~37 more groups, but tooling is making it possible to script routine AD maintenance, so this will be perfunctory by go-live.

A distant alternative in the future might be to approach instance management differently using "containerization." Supposedly, this emerging best-practice in "DevOps" amounts to a different paradigm, in which whole environments of interrelated servers, virtual networks, and parallel data can be created in concert by script, and development and testing is done exclusively in these parallel virtual environments. With containerization, prod AD would no longer need to contain OUs and members for other instances -- finally, something is much easier. Prod itself becomes a containerized set of resources and going live with a change can be accomplished by selectively restoring over changed prod entities.

No matter what the future holds, there is a proven need today for a 3-tier development environment in professional IS work. When the work is as complex as it is here -- with AD playing roles in domain authentication/security, software installation, and file server -- there is a special need for dedicated OUs, test users and extra computers (virtual or otherwise). Just as you should never have [prod] users in dev and never have test users in prod, you should never have dev users and/or do development in test. Not having the environment that we need exaggerates risks and makes the migration tasks unnecessarily complicated. Happily, these intricacies should calm down a lot when the migration is over and the bulk of the apps can return to normal maintenance.

Users Collection

Ideally, it would be simpler to start with a population of users who only had a few isolated Access 97 applications, and then uninstall Access 97 entirely before installing Access 2016 for those users. However, the pattern of installation is that there are a substantial number of apps that are widely installed among the ~260 accounts; there does not seem to be a way to avoid having both versions of Access for a substantial cohort of users.

One concern is that the migration itself will balloon the incidence of computers with multiple Access versions – basically exaggerating the scope of potential problems. A possible approach to shrinking the amount of users might be to exclude read-only users from the transactional database applications (and the migration?), and accommodate them outside of the applications with parameterized SSRS reports that allow them to gain access to the same data they currently use. However, this would not necessarily make the implementation easier, so it seems like something to consider after the migration, unless there is a critical need to shrink users at the expense of enlarging the scope of the projects in other areas.

Application Migration Planning

To complete the migration, a set of tools has been created in three parts: Application Inventory, Access 97 Entity Storage, Access 2016 Entity Generator. All three share a common SQL Server database which will probably be used from dev throughout the migration timeline: [devdbaot].[MigrateDMVa97].

Application Inventory

The Application Inventory establishes a dataset (primary table = tblApplication) representing relevant facts about migrating applications. The elements of the basic data model can be grouped into two types of relationships, which are understood from the perspective of a legacy application:

Collections (1-to-many)

Files - collections of files (among them, each app has a front-end/back-end pair of Access files containing assets known as "entities")
Users - collections of MDW group/users and/or AD group/members
Entities - tabledefs, querydefs, forms, reports, macros, and modules

Properties (many-to-1)

Disposition - Migrate to Access 2016, Migrate to Sharepoint, Migrate to VB.Net, Migrate to SaaS, Archive Data Only
Instance - Production Access 97, Development Access 2016, Test Access 2016, Production Access 2016

The Six Stages of Each Application's Migration

Stage 1: Access 97 Entity Storage

A set of tools is introduced into an offline copy of the application, which store in [devdbaot].[MigrateDMVa97] all essential facts about the six Access entities in the files collection of the app. Typically, relevant files consist of a back-end "database" containing only tabledefs, and a front-end "app" containing the other five entities: querydefs, forms, reports, macros, and modules. Other times the files collection has more Access members (aka “siblings”) and/or a “view” variation of legacy app whose entities will also be coalesced into a single app.

Stage 2: Access 2016 Entity Generation

A set of tools staged in a "blank project" are used to read the data from [devdbaot].[MigrateDMVa97] and generate SQL Server tables and all migrating Access entities in a new single front-end "app" containing: tabledefs (linked to SQL Server), querydefs, forms, reports, macros, and modules.

Stage 3: Dev Server Setup (SQL Server and AD)

Scripts upload the data from Access 97 to the Development Access 2016 instance of the SQL Server database. Alter table scripts reestablish auto-numbering IDs for fields that have imported legacy IDs. AD groups/users are established for dev.

Stage 4: Testing

SQL Server and AD are set up and administrated for the application's test instance. Fresh data is migrated from legacy prod to test SQL Server database. Acceptance testing is performed.

Stage 5: Production Access 2016 (Go-live)

SQL Server and AD are set up and administrated for the application's prod instance. Fresh data is migrated from legacy prod to new prod SQL Server database.

Stage 6: Recovery

Dev instances are recycled. Test instances are retained for maintenance, while test AD membership is cleared. Legacy prod fileserver assets are recycled. Legacy prod AD assets are removed.

Project Objectives & Success Criteria

#	Objective	Success Criteria
1	Create an inventory of applications in Access 97; identify a disposition for each.	All Access 97 applications identified. No apps remain “pending”.
2	Implement domain authentication by designing future AD security (groups/members) that follows legacy design, supports migration timeline, and respects standards of service for user admin.	AD/SQL Server design created with tools/procedures to allow rapid changes to AD group/membership.
3	Implement future ART installation process to allow Access 2016 to be co-installed with Access 97 and minimizes conflicts, managed by AD security groups/members, supports migration timeline, and respects standards of service for user admin.	ART installation process tested and implemented, including tools/procedures to allow rapid changes to AD group/membership.
4	Establish future app deployment process so database users can exist who do not have app, managed by AD groups/members, supports migration timeline, and respects standards of service for user admin.	AD design created to deploy apps (separate from security groups/members), with tools/procedures to allow rapid deployment of apps.
5	Designated apps migrate from Access 97 (FE/BE) to Access 2016 app/SQL Server database.	Objective completion.
6	Data from designated apps is archived in SQL Server and apps are taken offline.	Objective completion.
7	SQL Server data is available to designated stakeholders who have previously not had access to it.	Enhanced reporting capability established.
8	Migration of apps in legacy Access 97 portfolio to any other platforms, e.g. Sharepoint, VB.Net, and SaaS is eased/supported.	Migrations eased/supported. (UNKNOWN timeline).
9	Access 97 is uninstalled across DMV.	Last Access 97 software uninstalled after final Access 97 application migrated. (UNKNOWN timeline).
10	Prepare/retain SQL Server, AD assets, and tooling for ongoing support.	Retain dev and test SQL Server instances and complementary AD groups for future maintenance/testing. Retain application inventory application and database for administration as needed.

Project Scope

In Scope:

- Evaluating DMV Access 97 applications and determining disposition
- Implementing domain authentication to establish possibility for non-repudiation
- Exploring options to find best design for app deployment and packaging
- Designing/implementing app security in AD/SQL Server following current table permissions
- Enhancing app and table design to allow logging of login, and app version
- Making common splash screen with validation and error reporting to ease installation/troubleshooting
- Converting Access 97 applications to Access 2016/SQL Server
- Migrating data from "archived" applications to SQL Server
- Reorienting data/process stakeholders (e.g. analysts, managers, etc.) to SQL Server
- Easing/supporting app migrations to other platforms (e.g. SaaS, SharePoint, VS.Net)
- Removing all A97 software installs (no applications remain in Access 97)
- Establishing updated procedures and standards of service for administration and development

Out of Scope:

- Plan and execute migrations of Access 97 apps designated to migrate to other development Sharepoint, VB.Net, and SaaS
- Re-architect table/data design to normalize data that are repeated or overlap between databases.
- Evaluating/reevaluating data in Access portfolio in light of system of record beyond

Project Milestones, Deliverables & Schedule

Milestone/Deliverable	Target Delivery Date or Range
Phase 1: Planning	
Complete Access 97 application inventory	7/5/2017
Complete legacy AD design analysis	7/27/2017
Implement AD datasets for domain authentication, ART installation, and app deployment; demonstrate tooling for AD maintenance script generation	7/27/2017
Establish future app deployment process including Access 97 runtime applications co-exist with access 2016 applications	8/15/2017
Finalize charter	8/31/17
Platform testing completed	9/1/2017

Phase 2: Staged Migrations (x25)	1 per week average
Stage 1: Access 97 Entity Storage	
Stage 2: Access 2016 Entity Generation	
Stage 3: Dev Server Setup (SQL Server and AD)	
Stage 4: Testing	
Stage 5: Production Access 2016 (Go-live)	
Stage 6: Recovery	
Phase 2 Complete: All designated applications migrated to Access 2016 in production	3/1/18
Phase 3: Data-only Migrations	
All designated data archived in SQL Server.	4/1/18
SQL Server data is available to designated stakeholders who have previously not had access to it. by March 30, 2018	6/1/18
Prepare/retain SQL Server, AD assets, and tooling for ongoing support.	7/1/18
Phase 4: Follow-up	
Participate in migrations for apps designated to other platforms	(unknown timeline)
Access 97 is uninstalled across DMV.	(unknown timeline)
Project End Date	**(unknown timeline)**

Stakeholders

Stakeholder Group	Impact
Operations	They will be testing and running applications on

	Access 2016, which will have new and updated features from what they are currently using. So there will be a slight learning curve as they are understanding the learning the new access application.
Enforcement & Safety	They will be testing and running applications on Access 2016, which will have new and updated features from what they are currently using. So there will be a slight learning curve as they are understanding the learning the new access application.
Application Development	Maintain, design and support Access 2016 applications for users
PC Support	Deploy what application support staff develop
IT Infrastructure	Create and populate AD security groups

Estimated Project Cost

List the estimated project costs below. Add lines as needed for other types of costs that aren't listed in the table. Use the project's approved ABC form for reference, but be sure the table below includes any new or updated cost information obtained since the project's ABC form was approved.

Description	One-Time Cost	Annual Reoccurring Cost
Configuration/Installation/Implementation	$	$
Hardware	$	$
Equipment	$	$
Software/Licenses	$	$
State Labor	$	$
Project Management (contracted services only)	$	$
Other Professional Services	$	$
Service Level Agreement/On-going Maintenance Costs	$	$
Hosting Provider	$	$
	$	$
	$	$
Totals	**$**	**$**

Assumptions, Dependencies & Constraints

#	Type (Assumption, Dependency or Constraint)	Description
1		
2		
3		
4		
5		

Project Risks

Project risks are characteristics, circumstances, or features of the project environment that may have an adverse effect on the project or the quality of its deliverables. Identify known risks below, the impact and probability of occurrence, and the plan (i.e., the specific activities to perform) to eliminate or mitigate the risk.

#	Risk Description	Impact (H/M/L)	Probability (H/M/L)	Risk Plan
1	Resources	H	M	
2				
3				
4				
5				

Approvals

Role	Name and Title	Signature	Date
Sponsor	Jennifer Pittsley		
Application Development	David Pepper		
Project Manager	Kelly Nolan		
PC Support	Jason Boyd		
IT Manager	Jim Wood		