|  |  |
| --- | --- |
| NRCScolor[1] | **United States Department of Agriculture**  **Natural Resources Conservation Service**  **NRCS PSO Database Guidance**  Application Databases  January 11, 2018    UNITED STATES DEPARTMENT OF AGRICULTURE  Natural Resources Conservation Service  Production Systems Operations  2150 Centre Avenue, Building A  Fort Collins, CO 80526  USDA2color |

Table of Contents

[1.0 Introduction 4](#_Toc503967123)

[2.0 Official Documents 4](#_Toc503967124)

[2.1 Standards Locations 4](#_Toc503967125)

[2.1.1 NRCS-Deployments 4](#_Toc503967126)

[2.2 Microsoft Database Administration Guide 4](#_Toc503967127)

[3.0 Design Principles 4](#_Toc503967128)

[3.1 General Principles 4](#_Toc503967129)

[3.2 Application Definition 5](#_Toc503967130)

[3.2.1 Business Rules 5](#_Toc503967131)

[3.2.2 Data 5](#_Toc503967132)

[3.2.3 Interfaces 6](#_Toc503967133)

[3.2.4 Quality 6](#_Toc503967134)

[3.3 Database Architecture 6](#_Toc503967135)

[3.3.1 Database Naming 6](#_Toc503967136)

[3.3.2 Underlying Structure 6](#_Toc503967137)

[3.3.3 Database Schemas 7](#_Toc503967138)

[3.3.4 Cloud Ready 7](#_Toc503967139)

[3.4 Database Content 8](#_Toc503967140)

[3.4.1 Relational Design, Data Dictionary 8](#_Toc503967141)

[3.4.2 Constraints 8](#_Toc503967142)

[3.4.3 Indexes 8](#_Toc503967143)

[3.4.4 Aging, Sliding Window 9](#_Toc503967144)

[3.4.5 File Groups, Sets of Data 10](#_Toc503967145)

[3.4.6 Spatial 10](#_Toc503967146)

[4.0 Coding Principles 11](#_Toc503967147)

[4.1 General 11](#_Toc503967148)

[4.1.1 Source Control 11](#_Toc503967149)

[4.1.2 Transactions 12](#_Toc503967150)

[4.1.3 Error Handling 12](#_Toc503967151)

[4.2 SQL Server 12](#_Toc503967152)

[4.2.1 Efficiency 12](#_Toc503967153)

[4.2.2 Join Syntax 12](#_Toc503967154)

[4.2.3 UNION 13](#_Toc503967155)

[4.2.4 Hints (NOLOCK) 13](#_Toc503967156)

[4.2.5 Stored Procedures and Functions 13](#_Toc503967157)

[4.2.6 Cursors 14](#_Toc503967158)

[4.2.7 Triggers 14](#_Toc503967159)

[4.2.8 CLR 14](#_Toc503967160)

[4.2.9 Deadlocks 14](#_Toc503967161)

[5.0 ETL \ Batch 15](#_Toc503967162)

[5.1 SQL Agent jobs 15](#_Toc503967163)

[5.2 Marting 15](#_Toc503967164)

[5.3 SSIS 16](#_Toc503967165)

[5.4 Merge Statement 16](#_Toc503967166)

[5.5 Change Data Capture 16](#_Toc503967167)

[6.0 Development Environments 16](#_Toc503967168)

[6.1 Internal/External IP’s 16](#_Toc503967169)

[6.1.1 USDA vs. NITC HA 17](#_Toc503967170)

[6.2 Multi-Instance per host approach 17](#_Toc503967171)

[7.0 Security Principles 17](#_Toc503967172)

[7.1 Identity/Traceability 17](#_Toc503967173)

[7.2 Least Privileges 18](#_Toc503967174)

[7.3 Lockdown Scripts 18](#_Toc503967175)

[7.4 Grants 18](#_Toc503967176)

[7.5 DBA Prod permissions 18](#_Toc503967177)

[7.6 NRCS login synchronization 19](#_Toc503967178)

[8.0 “API” Cross Application Sharing 19](#_Toc503967179)

[8.1 Web Services 19](#_Toc503967180)

[8.2 Replication 19](#_Toc503967181)

[8.3 Three Part object reference 20](#_Toc503967182)

[8.4 Four Part object reference 20](#_Toc503967183)

[8.5 Direct Read connection 20](#_Toc503967184)

[8.6 Snapshot 21](#_Toc503967185)

[8.7 Log Shipping 21](#_Toc503967186)

[9.0 Resources, Performance, and Scaling 22](#_Toc503967187)

[9.1 Resources 22](#_Toc503967188)

[9.2 Performance 23](#_Toc503967189)

[9.3 Scaling 23](#_Toc503967190)

# Introduction

This document lists accepted database practices within the NRCS. This summarizes information that is relevant to database design and implementation and should be followed unless business needs and technical considerations require a variance.

# Official Documents

Follow NRCS standards and guidelines documents. This section should be updated as standards are published and clarified.

## Standards Locations

### NRCS-Deployments

This location shows requirements that are currently referenced in NRCS Production Systems Operations procedures.

NRCS-Deployments>Documents>[# Standards, Procedures, and Best Practices](https://colab.sc.egov.usda.gov/cb/dir/96683)

## Microsoft Database Administration Guide

This shows operational aspects to server/instance/database configuration:



[Microsoft\_Database\_AdministrationGuide\_V5.0.docx](https://colab.sc.egov.usda.gov/cb/displayDocument/Microsoft_Database_AdministrationGuide_V4.0.docx?doc_id=121222)

# Design Principles

## General Principles

A good design is based on detailed requirements. The designers should understand the business needs beyond the requirements. The designer should work with the business to rank the requirements and understand which are likely to change.

Application architecture describes all of the technologies and interfaces used within the application.

* NRCS recommends using standard technologies and currently supported software versions. Applications that choose non-standard software and technology assume the burden of acquiring and integrating these into the NRCS operating environment. Use of new technology starts at the lowest maturity level, so it will require addition time and expense to mature.
* Projects should list each technology and interface used in the architecture for use in their risk analysis and timeline.
* The actual design should include an overall description of how the application works and interacts with other systems. The overall description should be clear enough for all people involved to understand how each task fits with the whole.
* The design should include details on every component needed to meet the requirements. General principles:
* Clear enough for analysts to review and track how the design meets the requirements and use cases defined by the business
* Balance the design for use by application, database, report, and other developers
* Contain highly cohesive and loosely coupled modules
* Efficient approaches
* Operate with no (or minimal) manual intervention
* Include consistent error handling
* Handle expected outages from internal and external sources

## Application Definition

An application consists of business rules, data, and interfaces.

### Business Rules

The application design should identify what components enforce business logic. Having rules applied in reusable modules will promote re-use of code and simplify future changes to the rules. These reusable components should be placed based on scalability and where other modules that use them are placed.

The database can be used to enforce constraints as a secondary check. Constraints may be in the form of default values, reasonableness (range) checks, and foreign key constraints. Adding FK constraints to external data may not be possible or practical in some circumstances. The database should not be relied upon by the application code to reject invalid inputs, it should be viewed more as an additional safety.

### Data

The application is responsible for its data and data quality. It should limit the data stored to what is necessary and avoid PII (Personally identifiable or confidential information) as much as possible. It should enforce rules to ensure adequate data quality.

The design should recognize that there is a cost of managing data, and that large volumes of data can impact application performance. The application should include plans for deleting data that is no longer necessary or archiving it in other places where it will not impact current data.

### Interfaces

An interface is any input or outflow from the application. There are user interfaces like websites, and application interfaces like web services. Interfaces should be accounted for in the risk management plan of projects since they often involve new technologies, unknowns, and required cooperation of external groups.

Interfaces should be focused on bridging the flow of information and not specific business rules. They often will perform translations (ETL) and validation steps. They should include feedback mechanisms and monitoring to alert on issues and their frequency.

### Quality

The business expects a degree of reliability and quality that is often assumed. NRCS requires a certain level of error handling, robustness, and quality in all of its applications. A system should be built to facilitate future updates and be complete enough that manual intervention is rare. Unit testing should be included with every development task. These often unspoken requirements are to be included.

## Database Architecture

This refers to the interaction of databases with other components of the applications and structural recommendations that are not very application specific.

### Database Naming

Each database shall be uniquely named for backup file identification, and shall be easily associated with a project (owner.) Names should not be too generic like “Session” or “Log.” They should have application prefixes or some other descriptive identification.

This does not prohibit different versions of a database from having the same name. For instance, the log shipped copies of databases will share the same name, and the same name could exist for training, or perhaps be staged for a future upgrade. In these cases they all belong to the same application (owner) and are basically the same.

### Underlying Structure

The MSDBA Guide describes the ideal operational configuration of databases which is mostly beyond the scope of developers. The main area where developers can help is the definition of necessary file groups and changing the default from PRIMARY on any databases they create from scratch. Since SQL2005, Microsoft recommends keeping application data off Primary so databases can be partially online and support filegroup based backup/recovery options.

New NRCS databases should be created with a FG1 filegroup as the default with four data files. Additional files in groups of four are used to keep file sizes manageable and allow for better multi-threading of scans. Files within a filegroup should be the same size. Additional filegroups can be backed up and restored independently. Additional filegroups should be made for significantly different types of data like archive tables, binary/text, or other logical distinctions.

Operations will segregate log file storage from data storage per vendor recommendations to the extent allowed by hosting. Ops will also work to ensure proper alignment and blocking sizes which are generally 64k for data and 4k for logs. 8k was used for logs in the past, but 4k would allow for non-SQL files to be compressed on log drives.

Current NRCS development hardware and SAN test at about the same speed and throughput as production equipment now. The main difference is memory and in some cases production has additional CPU. If memory is over 8GB, the main performance difference is running FULL recovery mode without backing up translogs and not running index and statistic maintenance often enough.

### Database Schemas

Applications should consolidate core OLTP data into a single database. Applications traditionally have created separate databases to account for easier organization, performance, operational, and control.

Having separate databases makes it difficult to restore multiple databases and maintain the assumed consistency. Multiple database schemas should be used where database objects have different organizational and control requirements. Different filegroups can be used for separation in some performance improvement techniques.

Separate databases often make sense for logging and marting functions. These are outside of the core application transactions and generally have different update and management needs. These should not be directly referenced from other databases and could be hosted on other instances.

### Cloud Ready

NRCS does not know when application hosting will change, or the constraints imposed by different cloud environments. These are recommendations we feel will position applications to be prepared for some likely constraints.

1. Use single databases instead of multiple ones per application.
2. Avoid direct references to other databases using 3-part and 4-part naming.
3. Reduce any reliance on replication

Cloud vendors may use technology to move databases to different instances dynamically to adjust load. This could impact our ability to enforce rules that require databases to be co-located on the same instance or manage replication.

## Database Content

### Relational Design, Data Dictionary

OLTP databases should be generally designed in “normalized form” to prevent multiple copies of data and related update and consistency issues. Altering the relational design (de-normalization) to reduce joins especially where multiple large tables (over 10 million rows) are involved can be beneficial if consistency and update issues can be addressed.

Naming standards for tables and columns should be consistent within the application. A data dictionary should be included that describes the contents and purpose for each column. The same column name should have a consistent meaning across the entire application,

Data type recommendations for efficiency and performance:

1. Use char/varchar instead of nchar/nvarchar. The “n” types use double bytes and are only are useful to support certain Asian/Pacific languages.
2. Use variable length columns when length varies, and size is over 3 “characters.” Applications should not pad values with spaces unless required by a specific interface.
3. Decimal datatypes should use precision (first column) values of: 9, 19, 28, and 38. This allows the maximum size with the same amount of bytes. Even when business rules limit input to smaller numbers, the data type is often used for aggregation, so typically design for the highest sum the application will ever report.
4. Use Varchar (MAX) and Varbinary (MAX) instead of “blob” datatypes of text, binary, or image. The older types will be deprecated and don’t store data as consistently. If you must use the older datatypes, tables should be defined to use a special filegroup with this syntax: [ TEXTIMAGE\_ON { *filegroup* }

### Constraints

Constraints should be used to enforce default values where necessary. Foreign Key constraints should be used to ensure parent child relationships between business tables.

They may be used as an added safety measure to enforce reasonable data values. Foreign Key constraints may not work against domain tables depending on how they are updated. There are issues using FK constraints for replicated tables or tables that actually reside in other databases. Application should not rely on constraints to prevent invalid data entry.

### Indexes

**Clustered Index** - In SQL Server, a clustered index is used to sort the actual data records.

* Each table should have a clustered index at a minimum.
* Should be unique.
* The clustered index should be a sequentially increasing value which will allow data to be inserted without causing fragmentation.
* A single integer field will allow for optimal performance of the index tree. Keep indexes to the smallest number of bytes. For tables with over 1.2 billion rows, a bigint datatype may be necessary.

**Multi-Column Indexes** - For multi-column indexes, put the most selective “distinct” column first, unless reads will use other fields to filter more frequently. An example when the “exception” would be warranted is clustering by State, County, and Farm number. If reports and ranking frequently scan a specific county, this will limit I/O more of the time. A “general” example of this rule would be LastName, FirstName, and MiddleInitial. The last name is most specific.

**Fill Factor** - This determines how full data pages are normally. The default of 100 is good for data that does not change much, or keys that increment sequentially. For clustered indexes with very large rows (+4000 bytes) it should set to 100 as well because there is essentially only one record per page. For indexes that don’t meet the recommendations of 100, you can assume 80 to start and adjust based on the frequency of index re-orgs. The NRCS nightly reorg process will update any indexes that are over 8% fragmented. The fill factor should be low enough that reorgs occur less than weekly.

**INCLUDE Columns** - Non-clustered indexes can have included columns. If the application uses an index to find just 1-2 values for a given key value, adding those as include columns will save SQL Server from having to take an additional random access read to get the data record. Adding a column like an amount as part of the index proper, is less efficient because it expands the entire tree and sorting, not just the root level. Add INCLUDED COLUMNS sparingly when they can answer a question or help join needed tables without the added random access. This makes a second copy of these columns so it adds overhead for inserts and updates.

**Recommended Indexes** - System dynamic views, Performance Dashboard and DTA (tuning advisor) can suggest indexes to add and identify unused indexes. Reduce overhead by removing duplicate or near duplicate indexes. Multiple recommendations for an index with different sets of INCLUDED columns should not be blindly implemented. *In most cases adjusting a single index to have 2-3 included columns would have less overhead than three nearly identical indexes*.

### Aging, Sliding Window

Design should account for data disposition at the end of its useful life. For log aging, a sliding window partition scheme with associated job is an efficient method of removing aged data without locking the table or incurring extra overhead in the transaction log.

User data with no defined deletion date should still be archived in some manner to optimize access of active records. If archiving, we recommend a weekly process to a different filegroup to allow for more efficient backup cycles.

### File Groups, Sets of Data

Set FG1 as default and not on Primary at the database level. Create general database objects to be “ON [FG1}.” The explicit definition of FG1 should not be necessary when storing your object source code.

Some data objects to consider segregating into separate filegroups:

**Text/blob data** - Should be in a separate filegroup with TEXTIMAGE\_ON { *filegroup* } because the out-of-row storage is less efficient and prevents some normal operational optimizations to be done effectively.

**Archive/Static Data** - Sets that are no longer updated should be segregated into different filegroups so that backup frequencies could be altered if necessary.

**“Slow” data** - Current hosting does not support different speeds of storage, but for data that may not need normal response time, this would be a reason to define and use a separate filegroup.

**Indexes** - Testing has not shown separate file groups for indexes to have any measurable benefit in the current hosting environment. In situations where more physical control over the storage exists, moving non-clustered indexes to a different filegroup is a recognized technique to spread load. Having multiple files per filegroup has shown to yield better performance without having to manage separate filegroups.

**Other** - There are other situations where a separate filegroup is justified. One case is new development that wants to refresh “legacy” only objects. Using a filegroup only restore would allow current data to be copied in without disturbing newly added objects.

### Spatial

Historically spatial features have been implemented using dedicated databases with proprietary software. With the introduction of SQL Server “native” spatial support, there are new standards and recommendations:

1. Test for spatial index use. Developers must determine if the code they are executing for each spatial table is using the index. There is a compiler bug in SQL 2008, ‘08R2, and SQL2012. Certain ways of coding avoid this bug, or SQL hints can be used.

<http://www.sqlskills.com/BLOGS/BOBB/post/How-to-ensure-your-spatial-index-is-being-used.aspx>

Without this, 100-500 times the resources may be required to operate.

1. Put spatial data into separate tables. Experts in the field recommend this design. It allows for optimizing business tables for normal joining and reporting use, and allows for spatial optimization. Combining spatial data with large business tables generally limits optimization for only one of the competing uses.
2. Avoid Null geometry values. There is a known issue that SQL 2008 and SQL2008R2 will table-scan records that have null data in geometry fields when using the spatial index. This is supposed to be changed in SQL2012. This has not been explicitly tested by NRCS. Using separate tables for spatial data would prevent this from occurring because the “NULL” case is represented in a missing row and the missing row will not impact the spatial index performance.
3. Use integer keys over GUID’s.

Spatial index results require the random lookup of data rows using their primary key. Integer keys are generally half the storage of GUID’s which will allow for more records per page and various efficiencies. Integers that are sequentially added cause almost no fragmentation. GUID keys generate fragmentation during inserts and NRCS has little experience to judge the impact. Recently added records are often some of the most accessed, so this could be a key point to user performance.

1. Geographic alignment of Primary Clustered Key.

For optimal performance, use a clustered tabular key that groups geographic data close together. When a spatial index is used to find objects near a point, it will have pointers to data pages with the tabular values that are requested with the query. If the records in the search range are on the same set of pages, they would be in memory and avoid additional random seeks. A way to accomplish this for static data is to have a generic integer key/identity field and populate the table with an order by that would group geographic data close to each other. Hydrologic units, zip codes, Lat/Long substrings are methods to explore. The gains may only be noticeable on very large tables.

# Coding Principles

This describes coding techniques that are related to databases and their benefits. This does not define style guidelines, commenting, or other professional standards that are often applied to code reviews.

## General

General items may be applicable to database and non-database programming.

### Source Control

NRCS applications must maintain their source code in the agency approved repository. This should contain SQL schema, procedure, functions, SSIS, and other code. Since database code is rarely generated from scratch, the usage of source control is somewhat different. Projects should have a standard of how certain objects are stored. Specific “throw-away” implementation scripts will generally be separate from the main “trunk.” In the main trunk, each version of a table, procedure, or function should be relatable and able to be tagged with an application release.

Since procedures and functions require grants and can be implemented with either DROP/CREATE or ALTER syntax. Objects should always be managed with the application required grant statements.

SQL source code should generally include a <Return>GO<Return> at the end so that multiple scripts can be concatenated. Ordering of dependent objects is generally a manual effort or handled by tools like Redgate.

### Transactions

Programmers must anticipate problem cases like network or database outages. When adding a logical grouping of data that is stored on multiple tables, you must program this using begin transaction and commit transaction so that a partial set does not get recorded resulting in data fragments and other consistency issues. When nesting these, be especially careful.

### Error Handling

Good code should have error handling and error logging. Be aware of the impact on transaction handling and rollback that has been known to back out writes to error and log tables. Application should be able to handle situations when the database is unavailable.

## SQL Server

These are principles that are specific to SQL Server or database programming.

### Efficiency

Most database performance is focused on reducing I/O. This includes using indexes over table scans, efficient datatypes, and other ways to reduce I/O.

Specify only needed columns instead of using SELECT \* in production code. This reduces overall bytes, can prevent added random access reads when indexes could suffice. Being specific prevents errors common when columns are added to tables.

Write code that avoids table scans where possible. This includes tight WHERE clauses and proper indexes and statistics.

### Join Syntax

Use the JOIN and LEFT OUTER JOIN syntax that more explicitly defines the join behavior. Some behavior when the join is done using WHERE clauses can be interpreted differently. This also helps for readability by keeping filtering logic separate from joining logic.

The SQL server compiler is programmed to limit evaluation of various explain plans. By including larger tables first in the FROM structure, the SQL compiler will generally select a more optimal plan.

When building queries with joins, it is helpful to select record counts from the first table criteria (Joins commented out) and adding in (uncommenting) one table at a time to ensure no unexpected duplication or filtering of rows occurs.

Never use DISTINCT to resolve unexpected JOIN behavior, because hundreds of times the necessary I/O could be happening due to an unresolved Cartesian product.

### UNION

The results of SQL statements with Unions automatically filter identical rows in some cases. Ensure this behavior does not impact any aggregation or business logic. One way to resolve this is to add an unnecessary constant field that is different for each part of the UNION which will keep rows from different sections unique.

### Hints (NOLOCK)

SQL should generally not need hints for optimal performance. If there are performance issues, look at other sources first and use hints only as a last resort except for these exceptions:

WITH (NOLOCK) is recommended for read operations that contain a large result set and do not require the point in time accuracy. Without this, blocking can be a major issue for applications. Other SQL implementation have locking keywords to set the locking level without hints, but with Microsoft it was implemented with hints, but is a normal approved method.

WITH (MAXDOP=1) is commonly used when parallel executions plans are not performing well. We believe our standard instance setting of MAXDOP = 4, along with 4 equally sized data files will support normal parallel processing efficiently. For exceptions MAXDOP will turn off parallel processing.

INDEX hints are one of the spatial query options when the compiler chooses table scans over the spatial index. There are other options that avoid this, but the most common coding techniques could require this.

### Stored Procedures and Functions

The project team should decide how much of the SQL code should be managed in the application language vs. database objects. Ideally, the design should not require complex logic to be replicated on multiple languages.

Stored procedures have advantages and disadvantages. A stored procedure will store its first execution plan and save CPU cycles on later calls. If inputs can result in a single lookup in some cases and a large dataset in others, it is likely that the cached plan will give very poor performance in the opposite case. You should test extremes and either create different procedures or use the with Recompile option to ensure best performance.

Applications that code heavily with stored procedures will be more visible to DBA’s to determine impacts by potential schema changes. Complex SQL code should generally be assigned to more skilled SQL developers which also may weigh whether the design should use procedures or not.

NRCS Deployment policy now states that code changes in databases follow the same as other application code so the perceived advantage of faster implementation of database code no longer applies.

Functions return a single result and are good for simple lookups, formatting, and transformations. We caution against putting too many calculations or logic in functions that could otherwise be handled in application code because our model is to scale app servers, not database servers.

### Cursors

In most cases, SQL should be written without cursors. Cursors are not an integral part of the SQL language and perform poorly. In cases with few iterations they don’t have that big of an impact, but for thousands or millions of rows, every effort should be made to avoid them. They are also acceptable for rare processes like conversions where developer time savings is more important than conserving resources.

### Triggers

Triggers should be avoided in nearly all cases. They can cause performance problems, debugging confusion, and data integrity issues if they are lost. A few cases that make sense are auditing table changes or enforcing very complex data constraints. When triggers are required, a clear explanation of why other methods were not used will be required.

### CLR

CLR’s allow running non-SQL language code on the database server. For management and scalability, we do not recommend using CLR’s except for low-impact tasks like a batch import of data from a web service.

### Deadlocks

A deadlock happens when two processes are waiting on each other. These are rare. If you have multiple processes that alter tables, you may have some risk. The general guidance it to define the proper order to add/change/drop from multiple tables in transactions. If you go in defined order, you should avoid cases where two apps will be half way done but be blocked by the other. We see this in some cases with replication when something else is also writing or changing the same table. We recommend that nothing writes or alters tables populated via replication.

# ETL \ Batch

Much of the application operational support is related to scheduled tasks that are used for data aging, interfaces or populating data marts. This section defines coding standards related to these tasks. Section 8.0 show additional design considerations for API’s (interfaces.)

## SQL Agent jobs

Routine application database jobs to clear off aged data, run ETL tasks, generate reports or scan for problem cases. Jobs are named with meaningful prefixes so that SSIS and/or application jobs can be readily identified. Common prefixes include:

* SSIS Job, usually has *Appname* too
* Sync Usually stored procedures moving data to match.
* *Appname Application specific jobs*
* Snap Create or drop Snapshots
* Dbbak, ADMIN\_ PSO maintenance related
* LS Log shipping copy or restores

There may be additional standards to help define and track jobs for better overall management and tracking.

## Marting

Marting is a term where application (OLTP) data is copied into a separate database for reporting and aggregation purposes. The purpose of a separate database is to prevent uses with high I/O or CPU from impacting the application by segregating it onto different equipment. It also frequently combines data from multiple sources to be viewed together as a whole.

Best practices include:

* Processing updates via delta (changes) instead of full refreshes
* Use snapshots of data sources to ensure a clear update time and prevent reads from locking source database resources.
* Storing data in more de-normalized and star schema forms that are more responsive to analysis queries.
* Be judicious with storing multiple copies of the data. Only use “reporting” tables when performance requires them, or the point-in-time must be kept consistent. For example, store results if the business wants a quarterly report to have the same values indefinitely as a point-in-time snapshot.
* Adjusting backups so they avoid update periods, and don’t run on days when the data doesn’t change.
* Never drop tables as part of the ETL. ETL process should not need to perform DDL commands. Dropping and recreating tables will invalidate optimizations made by the SQL engine such as adding statistics.
* Batch processes that refresh table contents should consider a final step to update statistics or index maintenance to ensure the database engine will access them efficiently.

## SSIS

There are several initial standards to SSIS that are outdated. The current standards are to use SQL2016 SSIS, use file based deployments so that configurations are kept and not re-entered into the database every time there is a change. They go on the main data drive of the instance under the \APP\appname directory. A more current SSIS standard is to use the SSIS Catalog, which deploys .ispacs files into our centralized database instance NRCPADM2.

Future standards will likely store code in databases and log historical run times, errors, etc.

## Merge Statement

For data that is located on the same instance or database, a merge statement can reliably change a destination table to match a source, updating only changes. The changes occur 100% in the core SQL language and can be dozens of times faster than SSIS or other “cursor” oriented upload processes.

This does not work with non SQL data sources, or between separate database instances. To leverage this 2008 technology, the source data must be brought local via another method first.

## Change Data Capture

Change Data Capture will capture changes to base application tables and store them for a set period of time. This technology is recommended for auditing and marting requirements. There are operational considerations that need to be clearly communicated to prevent loss of historical data. These should be clearly represented in operational notes.

# Development Environments

The development environments are the responsibility of development. This section describes some common issues and solutions.

## Internal/External IP’s

NITC HA (Dev, PPD & Prod) were implemented with internal IP’s that are not accessible from the outside. This prevented a compromised host from divulging information that could be used to exploit other servers from the outside.

The general rule of thumb is that to access any server or instance from USDA or external data centers, you need to use the 196. IP address. For server to server access within HA-DEV, or within HA-Prod, use the 10. Internal IP address.

### USDA vs. NITC HA

Development laptops in the office (USDA) environment cannot simply connect to server names in the NITC HA. The servers in NITC HA do not exist in the USDA catalog. To access servers remotely, you need to use IP’s, host file entries, or configure SQL Server Alias so the name will resolve to the external domain resource.

For fileshares, Windows2012 Clusters do a check to ensure your destination matches the instance name, so it requires fileshare references to use the instance name. If you are mapping between domains, this means you must have the host file entry configured so the “external” name will resolve.

## Multi-Instance per host approach

In the past teams had a pair of VM’s configured for dev and test instances for their databases. Databases did not perform well on over-subscribed VM’s. Unused space was lost, and CPU’s were limited to 2. For projects needing more than two environments had to store databases with different names and adjust connection strings and three-part and four-part names which was error prone and labor intensive.

The new standard allocated 3CPU hosts that could support multiple SQL instances. Teams were free to have more than two environments without having to rename the databases. The entire host resources are shareable between the instances, and the NRCS was charged less than it was for multiple VM’s.

Each instance in NITC HA Dev should have a unique name for clarity, and they generally have an application or group prefix and either an environment suffix like \_Dev or \_Test or a generic number or letter to differentiate the instances. Applications connect to a named instance just like they do in production with an IP, port. Be sure to use 10. IP’s for server to server and 196. IP’s for external access.

# Security Principles

Operational practices are not described here, but some of the code specific items and design principles are listed for consideration in coding and design.

## Identity/Traceability

There should be no sharing of logins between people or applications. Any access from USDA to production **must** use a person’s p account. SQL logins should only be used by a single application and clearly identified as belonging to a specific application suite.

We prefer a single privileged and non-privileged account per application, but if there are features that may have unusual usage characteristics or exposure, then separate logins will facilitate tracking that feature’s usage and performance characteristics.

## Least Privileges

A general security principle is to grant only privileges needed to perform assign duties and work (people and applications.) NRCS generally simplifies these into:

ro “read only” Generally db\_reader, but could be to a specific schema. Could be to a specific schema and fcn’s and sp’s that don’t change data.

rw “read\write” Generally db\_reader and db\_writer but could be to a specific schema. Generally includes grants to fcn’s and sp’s.

p “public” Could represent ‘public’, access only to explicit grants

‘’ No suffix was owner or higher privileged login. May be used for sensitive data.

… Custom suffix for specific application delineation.

Application logins should not need owner or ddl\_admin privileges to operate, and these are generally a concern for auditors when they do.

More thought needs to be made into how sensitive information is protected. So we recommend isolating sensitive data into separate schemas to better restrict sensitive data from other modules of applications. This will need additional logins to restrict access but may be helpful with future audits.

## Lockdown Scripts

Prior scans have noted system procedures and other items that are applied to the SQL instance to reduce vulnerabilities. Applications should not use system stored procedures without clearly documenting the need and reviewing with current security recommendations. This also includes xp\_cmdshell usage which is phased out.

## Grants

Source code stored in subversion should have application specific grants included in the script. Updates to stored procedures that do a drop/create are notorious for dropping grants, so care must be taken to include the grant as part of the change control process.

Grants to procedures and functions that are given to ro logins (or developers) must not alter data. There are some logical exceptions that would include application logic that records viewing of records, error logins and such that is not actually conducting business within the application.

## DBA Prod permissions

We general allow DBAs these rights to view production information with their read-only accounts:

GRANT VIEW SERVER STATE TO [EDC\P\_XXXX\_PRD\_ReadOnly\_NRCS\_Role]

GRANT ALTER trace TO [EDC\P\_XXXX\_PRD\_ReadOnly\_NRCS\_Role]

GRANT VIEW ANY DATABASE TO [EDC\P\_XXXX\_PRD\_ReadOnly\_NRCS\_Role] -- [server level]

GRANT VIEW ANY DEFINITION TO [EDC\P\_XXXX\_PRD\_ReadOnly\_NRCS\_Role] -- [server level]

MSDB Specific grants:

EXEC sp\_addrolemember N'SQLAgentReaderRole', N' EDC\P\_XXXX\_PRD\_ReadOnly\_NRCS\_Role'

GO

## NRCS login synchronization

Technically, every sql login on every SQL instance is a separate unique identity, but NRCS treats every sql login as the same identity which should be traced to an individual or application.

# “API” Cross Application Sharing

Applications require a clear boundary of what it contains and controls. Access to data or services from other applications must be identified and adhere to basic rules to ensure each application can maintain its integrity. The concept of defining a clear API or boundary/handoff between apps is important for identifying and addressing impacts.

Here are the general recommendation of what type of API to use for different purposes

1. Web services are recommended for OLTP application to application interactions.
2. Replication/3-part/4-part naming is used for sharing lookup data. These should be interchangeable without code changes, and could be configured for optimal efficiency as resources permit. These are mainly valid when systems require complex SQL joins with tables managed by other applications.
3. Direct reads are generally applicable for SSIS and other Marting jobs that can read large sets of data. Snapshots are a variation on this that limits locking issues.
4. Log shipping should only be used for D.R. purposes and not read by other applications. Our log shipping standard will break server connections every half hour and can be “down” for long periods of time to resynchronize.

## Web Services

These are not database interfaces unless a CLR based job updates application data by reading them. This is the most preferred method of application to application interfaces.

## Replication

Replication is a clear API with publications from the owner and subscriptions for the remote users of data.

We recommend tables populated by subscriptions would:

* Not have constraints, triggers, or objects that had to be dropped/disabled and re-created when re-initializing.
* Nothing else should write to a subscription table. This has been known to cause deadlocks and impact replication.
* A new recommendation is to place subscription tables in their own schema so the data source is obvious in the database design and the above rules can be clearly communicated (or exceptions discussed.)

## Three Part object reference

Databases that reside on the same instance can reference objects in each other by using three part object naming. *Database.owner.object*

Normal application code or database code should NOT use three part naming of objects in different databases. Instead, when this is needed, the database should have “API” objects of either SYNYONYMS or specially prefixed views that map the remote database objects.

By defining “API” objects of SYNONYMS or VIEWS, if the remote database is moved, this can be replaced by Replication or Four Part object references by only changing the API objects.

## Four Part object reference

Four part object references are similar to three part object references, but they map to a database on a remote instance using a Linked Server. *LinkedServer.Database.owner.object*

Linked servers translate the request into OLEDB and incur additional overhead when used. Care should be taken to only use these for basic queries since some complex joining has been known in the past to require large volumes of data to be returned to complete the join.

Normal code should use two-part object naming, and any use of the four-part linked server access should be done through “API” objects of either SYNYONYMS or VIEWS.

For application linked servers, our standard is to define a generic name with \_Inst suffix like: CUST\_INST, MART\_INST… These are defined with the 3rd security option of:  For access to work, the same login must exist on the remote instance with the same password (as well as having valid permissions on the destination).

We do not use the linked servers with actual instance names so we can re-point applications when the destination moves by updating the linked server connection information instead of changing application code.

## Direct Read connection

There should be no cases where an external application writes directly to another application database. This breaks a fundamental boundary rule.

Direct database connections should be avoided in most circumstances for several reasons:

* Reading can sometimes block normal data access if done without (NOLOCK)
* Application databases may be upgraded and moved, so external readers add to testing and coordination.
* Applications may move to different sites, and WAN db connections are not approved for most uses.

There are cases where direct read access is pragmatic, especially in “marting” processes that transfer large amounts of data. Valid exceptions are usually ETL processes and not application db connections. There are also historical cases that were grandfathered, but not recommended.

When there is a valid exception to allow direct read connections to another application databases, these steps should be followed:

1. The application owning the data, must approve the external access.
2. The reading application, must use a different login clearly identifiable as an external login to read the data.
3. Information on what is being read should be documented and shared with the owning application.
4. The owning application can implement security grants to properly restrict access to data approved. Currently, most is implemented as db\_reader.

## Snapshot

A database snapshot will create a frozen virtual copy of an entire database which can be read without concerns of data changing or generating locks that would impact the live database. Since the snapshot is empty except for changes that occurs when it exists, it only takes seconds to start and performs very well.

While the snapshot exists, the old values of data pages are written as they are changed on the live database. There is over twice the I/O while the snapshot exists, and it will grow steadily based on the rate of changes. Therefore, we generally only take snapshots during the evening and not have them around during business hours.

## Log Shipping

Our log shipping method makes a full copy of a database, but must kick off users at: 25 and: 55 past each hour to apply updates. For some marting purposes, we pause the restores late overnight so that the data can be read without interruption. All permissions must be granted on the live source database.

Log shipped copies should not be used for application connection strings since log shipping can fall behind and may be unavailable for hours at a time when they require re-initialization.

# Resources, Performance, and Scaling

PSO helps NRCS understand and provision resources to meet the agency needs. There are not strict charging models to applications, but increased needs may require up-front funding when existing resources cannot meet the demands.

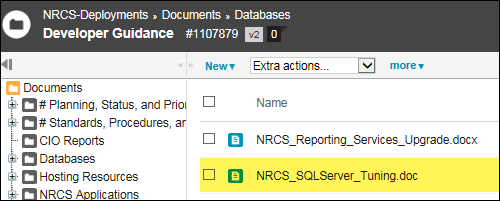
## Resources

NRCS is required to adhere to policies from USDA datacenters on what types of hardware, software and other computing equipment is used. There is limited NRCS control and changes to resources involves varying hard to predict timeframes.

We have worked to remove these obstacles from the development branch scope of concern, but projects need to cooperate from time to time. Steps to conserve resources and reduce risks to your projects are:

* Decommission resources that you are not using. Delete unused databases, files, accounts, etc. Take away configuration strings from app that are no longer used.
* Follow NRCS standard configurations. Use current versions of Microsoft SQL Server, IIS, .NET and incorporate this into project maintenance budgeting at least once every 2-3 years. We rarely have resources or depth of knowledge to support non-standard versions at the same professional level we aim for.
* Capture data lifecycle plans in requirements that include archiving or deletion of data after it is no longer active. Implement data cleanup.
* Reduce copies of data needed to fulfill application needs. Often views and other live queries can perform to expectations, so processes that make separate reporting tables and other copies of data for different uses may be wasteful.
* Reduce processing where possible. Often processing of only changes instead of full data sets will consume 1/100th or less the processing required to reprocess unchanged data.
* Compression. SQL has mature compression technologies and databases can also be compressed with NTFS compression in some circumstances. Compressing backups have saved about 70% of backup space and 70% of the backup time and similar gains can be achieved for some application needs if engineered correctly.

## Performance

There is a general performance document in: 

Applications need reasonable benchmarks for high frequency, and long running processes. These will help detect performance issues and highlight how changes impact users.

PSO monitors performance and works to identify and correct performance issues that are within the infrastructure and some that are noticeably outside of norms for a given database instance. PSO also works to ensure normal index maintenance, statistics and file sizes are set to optimal levels in production. Developers should ensure these are current in development when testing load and performance.

Applications should be aware and communicate changes to production that are expected to affect performance. PSO will work to accommodate legitimate needs. It will work to protect applications from affecting others and unexpected changes could result in a rollback or other negative impact to the application.

## Scaling

The scaling model within the NRCS production environment allows for relatively fast deployment of additional resources to application VM’s when required by an application. User driven program logic should ideally reside at this layer where that standard requires load balancing and we can react to load changes. The database is less able to scale. The options are moving databases between existing hosts to re-allocate available load or acquiring higher capacity hardware which can take up to a year to get into service. The database is likely the best place for batch logic, but logic that is driven by user interaction should be placed on the application VM layer as much as possible to allow for adequate scaling.