ActiveNet-Specific

SQL Server Database

Development Standards

# Preface

## Legal Notice

All original information, data, concepts and ideas presented in this document are the exclusive confidential and proprietary property of ACTIVE Network. Intellectual property law, copyright laws and international treaties protect the contents of this document.

No part of this document nor any of the information, data, concepts and ideas contained therein, may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any human or computer language in any form or by any means electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the expressed written permission of ACTIVE Network. Possession of this document does not constitute a transfer of any intellectual property rights or license rights.

All trademarks and registered trademarks are the property of their respective holders.

Copyright © 2016 ACTIVE Network. All rights reserved ACTIVE.

## Document History

The following revisions have been made to this document:

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Revision | Summary of Changes (Note section, item, and what has been changed) | Name |
| 1/13/2016 | 1 | Initial ANet version | David Christie |

# Introduction

In SharePoint, you will find the document “SQL Server Database Development Standards “ at this URL:

<https://activexperience.active.local/prodtechteams/gads/EDM/Standard%20Documents/EDM_SQLCodingStandards_v3.docx>

That document “outlines the guidelines that developers should follow when interacting with Microsoft SQL Server databases” at a general level which applies to all development teams at Active.

This document is an ActiveNet-specific addendum or commentary on the enterprise standards. There are a few reasons for this:

* Some of those guidelines don’t apply to ActiveNet. For example, ActiveNet has some naming conventions implemented through thousands of database columns which have a different naming convention. So if an ActiveNet developer were to just read the enterprise document, it might be confusing what to do: follow the existing patterns, switch to the new patterns for new columns, or even attempt to rename columns. In cases of such possible confusion, this document will describe the ActiveNet-specific practices, based on review with the DBE team.
* The enterprise document contains a limited number of code samples, but they are necessarily application-independent. This document will provide ActiveNet-specific code examples as they develop, as an aid to developers.

This document will be keyed to the enterprise document by including the section headings of the original document, then extracts of the original text and comments as appropriate. The original text will be italicized and indented.

# Development Standards for SQL Scripts

## All Data Access Performed Through Stored Procedures

All data access should be done through a Stored Procedure.

Activenet development should follow the practices recently put into place of creating stored procedures for all new database queries.

Also, whenever major maintenance is being performed on a section of code, requiring major retest, that is a good time to convert existing inline SQL into procs.

## All Procedures Created and Edited in SQL Server Management Studio

## All Procedures Follow a Naming Convention

The proper naming convention for stored procedures is as follows.

p\_TableName\_sel, upd, del, ins\_Freeformtext

Although the limited number of procedures already created do not follow this naming convention, all new procedures should.

## Avoid the Use of Cursors

## Do Not Use ‘Select Into’

## Do Not Use ‘Select \*’ In Queries

## Alias Column References for Clarity

If you are using more than one table for a query please alias the table and use the alias when referring to all fields as to make it clear from which table the field is from.

In many cases in which aliases are currently used in ActiveNet, the entire table name is used as the alias, which makes the code more verbose and doesn’t contribute to readability. Aliases should be short and suggestive, such as “receiptheaders rh” and “transactions tr”.

## Prefix Database Objects with Owner

All references to database objects should be prefixed with the schema to which the object belongs. In the vast majority of the cases this should be ‘dbo.’

For ActiveNet, this is indeed “dbo.”. All references to tables and other database objects should be prefixed with this schema.

## All Scripts Should be Commented

As we all know, uncommented code is difficult to maintain. Proper initial commenting of database code and the maintenance of those comments during any modifications is essential.

In addition to providing a header comment describing the interface, author, etc., any procedure of any complexity should be internally commented, just like code (which it is), so that the intention of the code can be quickly understood by the next person needing to read it. See an example in the section “Stored Procedure Coding Patterns” at the end of this document.

Use print command after every data change or after every schema change that your script does. This will help with finding problem in kit issue.

For clarity, this applies only to data changes (DML) which will be executed by the Kit Manager script. For example, data changes to accompany a new release or to populate new tables or columns from old, will be executed by KM and should have print statements. Print statements should not be used as a matter of course by DML executed within stored procedures.

## Business Logic Excluded from Stored Procedures

## All Procedures Should Return a Return Code

When the stored procedure returns a record set, the return code can't be read until the record set is closed.

When a stored procedure is called from a program and there is no error handling around it. The code will generate its own error on execution before it has a chance to check the return code. In this case you need a Try…Catch statement to execute the stored procedure in code so that you can check the return code.

For ActiveNet, the most common case of a proc is a series of SQL statements, with no special error handling. If there is a SQL failure, the proc immediately returns with an error, which we catch and log. No special error testing is needed; our code will Java catch the SQLException, log it, and do whatever error recovery it might do.

If there is no SQL error, any proc will by default return a return code of 0. This is also fine. There is no need to end the proc with a “RETURN (0)” statement.

For ActiveNet, the only case in which we should use a return value is if there are tests in the proc for error conditions which don’t cause a SQL error. For example, if the proc executes an UPDATE statement, which is expected to modify one or more records, but modifies none, this might be considered an error condition. In this case no SQLException will be raised in our code. Instead, the proc should return a status code like this:

-- If no records are updated, this is unexpected; exit now and return -2

if @@rowcount=0 begin

set nocount off

return -2

end

The Java code will save to check the return code; we have no current example of this, so we will have to add a Java example here. For reference, here’s how to get the return code in SSMS when executing a proc:

declare @r int

execute @r = myproc '1000000'

print @r

If a proc can return a status code, it must be documented in the proc header comment:

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

...

Returns: 0 on success, -2 if no records are updated

...

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

## Explicitly Define Security Access Using Database Roles

(No change in practice at the current time)

At the end of each script defining a stored procedure should exist a granting of execute privileges on the stored procedure to a specific role

TBD what the roles should be and how they are used. The current Kit Manager scripts generated by DBA have \*no\* GRANT statements at all. DBA will need to take the lead in this area; until there is a strategy, developers should not attempt to address this.

## Use SQL-92 Coding Standard

## Use With (NOLOCK) Hint (Dirty Read) Where Possible

In a high transaction database environment the use of the With (nolock) hint in a select query greatly improves performance by eliminating the requirement for SQL Server to maintain locks. This is only appropriate when a noncommitted (“dirty”) read operation is acceptable…

The ActiveNet database connection system currently ensures that reads are executed with isolation level read uncommitted, unless the developer explicitly requests a different isolation level. This eliminates the need for ANet developers to use the (nolock) hint in select queries.

## Do Not Use Exec(sqlstring) Method in Stored Procedures

Stored procedures run in the context of a SQL Server administrator and therefore the use of the Exec (sqlstring) method from within a stored procedure will violate security rules. It also makes it impossible for the SQL query optimizer to reuse compilation plans for database requests and therefore adds overhead of having to constantly perform SQL compilations.

Of the small number of existing procs, some are creating dynamic SQL and exec’ing it. These should be fixed at some time.

However, there are a few existing examples of how to solve problems which we traditionally solved with dynamic SQL. See the section “Stored Procedure Coding Patterns” at the end of this document for examples.

## Avoid Sub Queries/With Large IN or NOT IN Clauses

## Procedures returning one value use an OUTPUT Parameter

A common usage of this in ActiveNet is an insert proc returning an ID.

Example: CREDITCARDSARCHIVE\_INSERT

@id int output

as ...

-- Get ID of inserted record (if any) as return value

if (@@error <>0)

set @id=0

else

set @id=@@identity

## Procedures returning input variables

## Query validation

This is the standard syntax to validate the execution of a query in a procedure.

INSERT statement:

If @@Error <> 0

Begin

Set NoCount Off

Return -1

End

In ActiveNet, we will not do this. See comments in section 3.11. Unless your proc is going to implement special internal error handling for some SQL error, no special processing of SQL errors is necessary. The proc will immediately fail with an error,

## Index Creation

# Data compression

All new tables should be created with page compression

CREATE TABLE <tablename> ... WITH(DATA\_COMPRESSION=PAGE);

All new indexes should be created with page compression

CREATE INDEX <indexname> ... WITH(ONLINE=ON, MAXDOP=0, DATA\_COMPRESSION=PAGE);

**Do not do this!** Note that we have had problems using compression with our organizations. For SQL 2012, it is only available in the Enterprise edition, which our customers generally had not paid for. As a result, they were unable to restore their database backups. Based on this, we will continue without data compression for the time being.

# Database Object Design Standards

## Naming Conventions

### Table Naming Conventions

#### General (Data) Table Naming Conventions

For compound names, the component words are concatenated and the first letter of each word is capitalized. For example, “MessageText”.

Note that an underscore (“\_”) is allowable only in temporary or replicated tables. See recommendations below describing Temporary Table conventions.

Existing ActiveNet practice is that an underscore \*is\* used to make multiword names readable, instead of camel-case as suggested above. You should continue the current practice.

Names should not be longer than 30 characters.

Although there are many existing table and column names longer than 30 characters, please follow this convention for all new tables and columns.

#### Reference Table Naming Conventions

#### Replicated Table Naming Conventions

#### Temporary Table Naming Conventions

### Column Naming Conventions

A column name is unique in the database.

This is not currently the case, as there are many examples of columns named NAME or DESCRIPTION. However, for new columns, please follow this rule. For example, in a table like DASHBOARDS, you might call the column DASHBOARD\_NAME instead of NAME.

Note that according to the standards, foreign key names are an exception; they will normally have the same name as the primary key they reference. This is consistent with ActiveNet’s practice. So ACTIVITIES.ACTIVITY\_ID is a primary key, and TRANSACTIONS.ACTIVITY\_ID is a foreign key referencing it.

#### Identifier Columns

The name for an identifier is the table name (singular) concatenated with “ID”, the abbreviation for “identifier”. For example, “ApplicationID” is the identifier for the Applications table.

In existing practice, ActiveNet identifiers end with “\_ID”. Continue to follow this practice.

#### Foreign Key Columns

### Constraint Naming Conventions

In general, any new constraints should follow the enterprise naming convention,

#### Primary Key Constraint

#### Foreign Key Constraint

#### Unique Key Constraints

### Index Naming Conventions

In general, any new indexes should follow the enterprise naming convention,

### Common Abbreviation Naming Conventions

## Table and Column Design Considerations

### Data Types

Don’t use non-Unicode data types unless exceptional with special reason.

Because in existing practice all ActiveNet character columns are varchar (non-unicode), continue to use varchar for new character columns.

Avoid using ntext and image data types in new development work as they will be removed in a future version of Microsoft SQL Server and plan to modify applications that currently use them. Use nvarchar(max) and varbinary(max) instead.

For any new large text columns, use “varchar(max)” instead of “text” as was done in the past.

### Primary Keys

### Foreign Keys

Relationships should be logically defined through our naming convention as described above. They should be defined in Data Models. However, due to performance issues, we usually do not physically enforce foreign keys with constraints in the DB. Apps must ensure data integrity.

This is not the current ActiveNet model. Declarative referential integrity (DRI) is used on all foreign key relationships. Continue with this practice until further notice.

### Index

On tables that can be updated online, no more than 5 indices are allowed. This ensures the performance of the online systems.

Ultimately, the decision to create an index on a table will be made by a DBE/DBA.

In the current ActiveNet model, there are indexes on most FKs, so some transactional tables have many indices. Continue with this practice until further notice.

### Triggers

No triggers are currently used in ActiveNet. We should continue to use that practice unless there is a compelling need to do otherwise, reviewed by DBA.

#### Replacing triggers with declarative referential constraints

#### Replacing triggers with stored procedures

# Database Release Best Practices

## Database Environments

## Roles in the Release Process and Overview

## Database Release Process

### Project Development Releases

### Releases to QA

### Production Release

### Emergency Release

#### Hot fix

# DB - Prod. Approval Without QA

# Rollbacks

## Recovery criterion

## Data scripts

### Non Release Related

### Release Related

## SSIS packages

## Methodology Guidelines and the EDM Rollback Database

### Updates

### Inserts

### Deletes

# Miscellaneous Database Development Considerations

Every table in the database should have the column ModifiedDate (at least defined as SMALLDATETIME) that contains the date and time the row was initially inserted into the table or modified. The column should be defined as NOT NULL. ModifiedDate may have a default value of getdate() to populate the column during record insertion. ModifiedDate MUST be maintained by all procedures making a modification to any column in the row.

TBD: We certainly could add a column with a default to existing tables in just the KM scripts, but ensuring it is updated in all cases would be considerable amount of work. Open question is whether we should make this a standard for all new tables; possibly for existing tables when significant rework to classes is being done.

# Kit Manager

# Writing Your code : SQL SERVER Re-Runnability Standards:

# Roles and Responsibilities

# Setup

# Process Flow

# Stored Procedure Coding Patterns

## Comment SQL logic

In addition to providing a header comment describing the interface, author, etc., any procedure of any complexity should be internally commented, just like code (which it is), so that the intention of the code can be quickly understood by the next person needing to read it.

Example: SET\_CUSTOMER\_ENCRYPTED\_PASSWORDS

as ...

-- Update password values in the customer record

UPDATE CUSTOMERS SET IS\_ENC\_PASSWORD=-1,

PASSWORD1=case when LEN(@encrypted\_password1)=0 then password1

else @encrypted\_password1 end,

PASSWORD2=case when LEN(@encrypted\_password2)=0 then password2

else @encrypted\_password2 end

WHERE CUSTOMER\_ID=@customer\_id and IS\_ENC\_PASSWORD=0

if (@@ERROR=0 and @@ROWCOUNT>0)

BEGIN

-- Insert record into the CUSTOMERLOG table for password encryption

INSERT INTO CUSTOMERLOG

(CUSTOMER\_ID,SITE\_ID,WORKSTATION\_ID,SYSTEMUSER\_ID,DATESTAMP,

ITEMNAME,CHANGETYPE,ITEMTYPE,CHANGE\_DESCRIPTION)

VALUES (@customer\_id,@site\_id,@workstation\_id,@systemuser\_id,getdate(),

@item\_name,@change\_type,@item\_type,@change\_description)

END

## Passing an ID list into a proc for use in a WHERE clause

Example: rpt\_GenerateAndRunCompanyListingReport

This proc is provided a parameter (@geographic\_area\_id which is a comma-delimited list of company\_id’s and geographic\_area\_id’s (ints). If that string is non-empty the code needed to include it in the where clause, without adding the string of IDs into the SQL and exec’ing it.

@company\_id varchar(max),

@geographic\_area\_id varchar(max)

as ...

where

-- Match company\_id to the ID list if provided

((len(@company\_id) = 0

and (COMPANIES.ParentCompany\_ID = 0 or COMPANIES.ParentCompany\_ID is null))

or (len(@company\_id) <> 0 and COMPANIES.COMPANY\_ID in

(select val from dbo.fn\_MVSplit(@company\_id))))

-- Match geographic\_id to the ID list if provided

and (len(@geographic\_area\_id) = 0

or (len(@geographic\_area\_id) <> 0 and COMPANIES.GEOGRAPHIC\_AREA\_ID in

(select val from dbo.fn\_MVSplit(@geographic\_area\_id))))

Note the use of the function fn\_MVSplit, which returns a table containing all the IDs in rows of a table which have a single int column “val”.

Also notice that this code currently violates 3.16 by using an IN clause instead of a JOIN; there may be a better way to do this, but the current code at least is compilable and doesn’t use exec.

This pattern is useful if a parameter which is list of IDs is provided from UI, or possibly from a variable list of enum values, such as a list of transactiontypes.

However, we have code now which executes a query, builds a list of IDs from the result set, then puts that list into an IN clause. Don’t do this; it’s way more efficient to put the query into the proc itself as a subquery or to populate a temp table.