November 6, 2010

DEPARTMENT OF BEHAVIORAL HEALTH AND DEVELOPMENTAL SERVICES

SQL Server Standards

Design and Development

Office of Information Technology Services

11/6/2010

This document contains design and development conventions and style guidelines that will ensure that SQL Server database objects will be consistent and superior quality. It includes general guidelines as well as discussions on naming, formatting, and use patterns.

I. Contents

Revision History

Date	Revision	Description	Author
11/07/2010	1.0	Initial Release	Wendy J. Cary

II. Database Design

A. Database Scope

Define the database scope before beginning modeling. Questions that must be answered to define the scope are:

1. What will be included in the database?
2. What will not be included?
3. Define the major use of the database (transaction processing, reporing, data warehouse)
4. Determine the scope of use (within an office, department wide, multiple agencies, external to the Commonwealth)
5. What other people or applications will be accessing the database? This is critical for security, user role and group definition planning, and table/view design.

Also, during database design, consideration must be given to items such as Integration Services (SSIS) and query optimization. Although not directly linked to the design process, the database design will impact the development and optimization of these objects.

B. Conceptual Data Model

The conceptual data model is a tool used to define initially the domain concepts and database objects with the project owner. This model is not concerned with the platform on which the database will reside nor the detailed logic describing relationships among database objects.

Requirements: There are no specific requirements for the approach to take in working with the conceptual data model. However, it should be as inclusive as possible to aid in defining the database scope prior to the creation of a logical data model.

C. Logical Data Model

The logical data model fully describes and defines all database objects and their relationships. It is independent of how the database will be implemented in a specific database management system. It consists of three major components:

1. Structure — This component consists of the entities (tables) in which the data will be stored.
2. Constraints — This component contains the rules that define what makes the data valid.
3. Rules — This component defines data values such as default values, allow nulls, etc.

Requirements:

a. Use Visio to diagram and document the model.
b. Define all entities (tables) including their definitions in extended properties.
c. Define the attributes (columns) of the entities including their definitions in extended properties.
d. Normalize the database to third normal form if the major use will be transaction processing. This will increase database performance.
e. De-normalize the database if the major use will be reporting or data warehousing. This will reduce the number of joins needed to retrieve data, thereby increasing performance.
f. Define all relationships between the entities, including cardinality (one-to-one, one-to-many, many-to-many, etc.).
g. Resolve all many-to-many relationships with linking tables.
h. Identify a primary key for each entity.
i. Identify all foreign keys.
j. Identify column attributes (data type, size, allow nulls, etc.).

D. Physical Data Model

The physical data model is the implementation of the logical data model for a particular database management system. To create SQL Server database objects, generate the DDL (data definition language) statements from the logical database structure that was developed using Visio. These statements will include the create statements for tables, columns, indexes, constraints, etc.

Requirements:

a. Generate all DDL statements directly from the Visio logical data model diagram.
b. Before creating the database object, check for its existence. If it does exist, delete it prior to executing the create statement.

III. Security

Requirements:

a. Permissions must be set at the minimum necessary to perform required functions.
b. Permissions must also be set at the role level, not individual user level.
c. Use views to prevent direct access to database tables.
d. Use stored procedures instead of embedded SQL statements for all database queries.
e. Do not use .ini files for database logon and connection information.
f. Do not use .config files for database logon information.
g. Encrypt passwords stored in database tables.

IV. Database Scripting and Promotion

Requirements:

a. All database objects must be scripted. The initial DDL statements must be generated from the Visio logical data model diagram. SQL Server or Red Gate SQL Doc must be used to generate scripts for any database modifications.
b. Current change management procedures must be followed whenever promoting database changes to test and production.

To deploy an SSIS package, do the following:

1. Use the Deployment Utility to specify a deployment output path, set Create Deployment Utility to "True," and set Allow Configuration Changes to "True."
2. Build the deployment package.
3. Copy the deployment directory and associated files to the SSIS folder for your deployment at \\Co02\ITSDev\Production Changes.

V. SSIS Packages

Requirements:

a. Create, save, and build the package using Business Intelligence Management Studio (BIDS).
b. Save all packages to the local file system (not a network server).
c. Give each package object a descriptive name (see Naming Conventions) and document its function at the time it is created.
d. Use the Script Task when a placeholder or dummy task is required and fully document why it was used.
e. Use the Grouping Container to group tasks in the development environment. This is for readability since this container has no effect on package execution.
f. Never change the package's code in Code View. Editing in this manner can introduce unexpected errors.
g. When creating a new package from an existing package, always generate a new package ID to avoid confusion between packages with the same ID.
h. Use the File System task for local file operations, not the FTP task.
i. Unlike DTS variables, SSIS variables have scope. Avoid naming locally scoped variables with the same name as globally scoped variables.
j. Set Generate Debugging Information to "off" when promoting a package to test or production.
k. When retrieving an SSIS package from VSS, do not use the merge option. This would corrupt the package's internal references to other objects within the package.

Warning: Saving Packages with Encryption

When an entire package is saved encrypted, remember that no one can log on with the credentials of the person who created the package; the entire package would be lost. Saving with one of the following options will allow the package to be recovered: Encrypt Sensitive Data with Password, Encrypt Sensitive Data with User Key, or Do Not Save Sensitive Data.

VI. Naming Standards

A. Databases

The database name must consist of the project acronym or name in all caps. For the development and test environments, add a descriptor (see below) for the environment. Do not use spaces or underscores in the name.

Examples:

1. Production CCS3
2. Test CCS3Test
3. Development CCS3Dev

B. Schemas

Schema names should be descriptive of the group of objects contained in it and should be all lower case. Because the schema should be specified when referencing any object in a script, keep the name short. Schemas can be helpful in securing data. For example if you have users who are not authorized to view HIPAA data, create a separate schema that contains HIPAA data and only give the appropriate users permissions to this schema.

C. Tables

Use names that are descriptive of the underlying data contained in the table. Table names must be plural. The name must consist of a prefix (lower case) plus the table name (mixed case). A linking table's name should contain the names of the two tables being linked. Do not use spaces or special characters in the name.

Prefixes:

1. Base Table tbl
2. Linking Table lnk
3. Lookup lkp

Examples:

1. tblAccounts
2. tblRoles

3. lnkAccountRoles
4. lkpGenders

D. Views

Use names that are descriptive of the underlying data contained in the view. View names should reflect the purpose of the view. The name must consist of a prefix (lower case) plus the associated table name (mixed case). Do not use spaces or special characters in the name.

Prefixes:

1.	View	vw

Examples:

1.	vwAccounts	Used when the view returns all values in a single table.
2.	vwNewAccounts	Used when returning a subset of values in a single table.
3.	vwAccountsRoles	Used when returning all values from two tables.
4.	vwSalesByQuarter	Used when returning summary data for reports.

E. Columns

Use names that are descriptive of the data contained in the column.

Rules:

1. The column name must be singular.
2. Do not use prefixes for column names.
3. Do construct a column name with a prepositional phase: instead of DateOfSale, use SaleDate or DateSold.
4. When the primary key is an identity or autonumber, it must be the singular of the table name with a suffix of ID (ex. AccountID).
5. Foreign keys must be named the same as the referenced column in the parent table unless that data is used multiple times in the same table. When used multiple times, name the column for the purpose it's being used for (ex. PersonID from tblPeople may be used in tblEmployees to reference who the employee is and to reference who that employee's manager is. In this case, tblEmployees may have both PersonID and ManagerID as foreign keys to the tblPeople table.)

Examples:

1. FirstName
2. DiscountPercent
3. UnitCost
4. DateOrdered

All tables must include the following four columns with these descriptions in the extended properties:

1. CreatedBy — Username of the person who inserted the record.
2. CreatedDate — Date and time the record was inserted.
3. ModifiedBy — Username of the person who last modified the record.
4. ModifiedDate — Date and time the record was last modified.

F. Indexes

The index name must include the column name to which it applies. This makes it easier to analyze execution plans generated by SQL Server. Remember that multiple non-clustered indexes are allowed on a single table.

Prefixes:

1. Clustered Index — PK (same as the primary key)
2. Non-Clustered Index — IX

Examples:

1. PKAccountID
2. IXLastName
3. IXEmail

G. Keys and Constraints

Constraint names must reflect the scope of the constraint and the name of the table to which the constraint applies. The scope includes checking for valid values, unique, primary, and foreign key constraints.

Prefixes and Naming Rules:

	Type	Prefix	Naming Rule
1.	Primary Key	PK	table name
2.	Foreign Key	FK	referencing table + referenced table
3.	Unique Key	UK	table name + column1 [+ column2 + …]
4.	Check Constraint	CK	table name + column1 [+ column2 + …]

Examples:

1. PKtblAccounts
2. FKtblAccountslkpGenders
3. UKtblAccountsEmail
4. CKtblAccountsZipCode

H. Stored Procedures

Customized system stored procedures must be created in the Master database (required by SQL Server). They must begin with the prefix "sp" followed by the descriptive name of the stored procedure.

User-created stored procedures have the user defined prefix, followed by the predominant action, followed by a descriptive name (see descriptors below).

Prefixes:

1.	Custom System	sp	NOTE: Follow the prefix with a descriptive name.
2.	User Defined	usp	NOTE: See list below for remaining naming parts.

Predominant Action and Descriptor:

	Predominant Action	Abbr	Descriptor
1.	Returns multiple rows	SEL	Plural of table name [+ subset descriptor]
2.	Returns single row	GET	Singular of table name
3.	Update	UPD	Singular of table name
4.	Delete	DEL	Singular of table name
5.	Insert	INS	Singular of table name
6.	Insert/update	SAV	Singular of table name
7.	Validate	VAL	Singular of table name
8.	Export	EXP	Business Process
9.	Import	IMP	Business Process
10.	Report-specific	RPT	Report name

Examples:

1. spGrantDELTALogin
2. uspSELAccountTypesActive
3. uspGETAccount
4. uspUPDAccount
5. uspINSAccount
6. uspEXPActivityLogAudit
7. uspIMPVDHReferrals
8. uspRPTAccountProfile

I. Triggers

SQL Server provides two types of triggers: DDL (data definition language) and DML (data manipulation language).

Prefixes:

1. DDL Triggers trd
2. DML Triggers trm

1) DML Triggers

There are two types of DML triggers: AFTER and INSTEAD OF. AFTER triggers can be specified only on tables and will be executed after the triggering item. INSTEAD OF triggers are executed in place of the usual triggering item. Multiple triggers can be created for each triggering action and a single trigger can be created for multiple triggering actions. Because of this flexibility in creating triggers, describe the business process instead of using a prefix for the triggering action.

DML Trigger Types:

1. After AFT
2. Instead Of INT

Examples:

1. trmAFTtblAccountsAudit
2. trmINTtblAccountsInsert

2) DDL Triggers

DDL triggers are new to SQL Server 2005. They enable us to perform actions based on changes to database objects and logons. No naming standards have been developed for DDL triggers yet. If you need to create this type of trigger, discuss naming with the DBA.

J. User Defined Functions

User defined function names must describe the results the function returns.

Prefixes:

1. User defined function udf

Examples:

1. udfIsAccountActive
2. udfIsAccountInRole

K. Rules

Rules are a backward compatibility feature. CREATE RULE will be removed in future versions of SQL Server. Avoid using CREATE RULE in new development work and plan to modify applications that currently use it. Instead, use check constraints. Check constraints replace this functionality and are the preferred, standard way to restrict values in a column.

L. Defaults

Defaults are a backward compatibility feature. CREATE DEFAULT will be removed in future versions of SQL Server. Avoid using CREATE DEFAULT in new development work and plan to modify applications that currently use it. Instead, use default definitions created using the DEFAULT keyword of ALTER TABLE or CREATE TABLE.

M. User Defined Data Types

User defined data type names must include the name of the column to which the data type applies.

Prefixes:

1. User defined type udt

Examples:

1. udtEmail
2. udtUsername
3. udtZipCode

IMPORTANT NOTE: A user-defined type is implemented through a class of an assembly in the Microsoft .NET Framework common language runtime (CLR). For SQL Server 2005 to bind a user-defined type to its implementation, the CLR assembly that contains the implementation of the type must first be registered in SQL Server by using CREATE ASSEMBLY.

N. SSIS Packages

SSIS Package names must be prefixed with the name of the database followed by the business process.

Examples:

1. AVPMCreateDiagnosisCodeTable
2. OLISDevLoadProviderData
3. FIMSCopyDataForReports

VII. Data Integrity

There are three types of data integrity. All three must be used to ensure the integrity of database information.

A. Domain Integrity

Domain integrity places restrictions upon values that are valid for data in a table's columns. Enforce domain integrity using data types, allow null attributes, and check constraints.

B. Entity Integrity

Entity integrity restricts all rows in a table to being unique. Enforce entity integrity using primary keys.

C. Referential Integrity

Referential integrity requires that data in one table retain its relationship with data in another table. Enforce referential integrity using foreign keys.

Declarative or procedural methods may be used to enforce data integrity. Declarative methods include check constraints and defaults. Procedural methods include stored procedures, triggers, and application code.

NOTE: Declarative methods are preferred because they are a part of the database, are easy to implement and manage, and have low overhead. Procedural methods impose a higher overhead cost on the database and are more complex to implement and maintain. They are, however, the preferred method (and in many cases the only method) to implement complex business rules.

VIII. Keys and Constraints

A. Primary Keys

A table can only have one primary key. The primary key consists of one or more columns that uniquely identify each row within a table. Nulls are not allowed. By default, a clustered index is created on the primary key.

Required:

a. Set a primary key for each table.
b. Do not create multi-column primary keys. Instead, create a surrogate key by using the identity property to generate a unique ID for each row. Then, create a unique index on the columns that uniquely identify a row of data. The exception would be linking tables where the only two columns in the table are foreign keys.
c. If the primary key contains more than one column (not recommended), the first column of the key should be the most unique element or the column that will be used in most queries.

B. Foreign Keys and Referential Integrity

Foreign keys reference columns in another table, thereby establishing a relationship between the two tables. The referenced column must be a primary key. The referencing column and referenced column must be of the same datatype.

Remember to set the ON DELETE and ON UPDATE properties on a foreign key to properly enforce referential integrity. Available options are: no action, cascade, set null, and set default. Use the Cascade option to cause update and delete actions that are initiated against a referenced table to "cascade" down to the referencing table.

C. Unique Keys

Unique keys are similar to primary keys in that they uniquely identify a row in a table. However, unique keys will allow one (and only one) row to contain a null value in the column(s) to which the key applies. Use only if a column or multiple columns other than the primary key must be unique. Where appropriate, create using the NOT NULL clause to avoid the problems associated with having one row being able to have the column's value set to null.

D. Check Constraints

Check constraints impose restrictions on what can be entered into a column. The expression for a check constraint must result in a Boolean value that must evaluate to "True." Use check constraints instead of the defunct Rules.

E. Computed Columns

Computed columns are virtual columns with their values being computed on demand. There is no physical column in the database to store the computed value. They avoid the problem of computed column values and their base values becoming out of synch. The value of a computed column is the result of an expression that uses column values from the same table or literal values. An index can be created on a computed column.

Restrictions:

1. Cannot be used as a foreign key.
2. Cannot be used as a default.
3. Cannot use a sub query.
4. Values must come from one table.

IX. Script Writing Standards

A. Structure

Requirements:

a. All create scripts must start with a DROP statement to ensure that another object of that name does not exist. Be sure to include the IF EXISTS clause to prevent errors if the object does not exist.

b. Do not include a USING statement. Instead, ensure you are executing the script against the correct database. This prevents the promotion manager from changing the database name when promoting the script.
c. All scripts must include a comment header after the DROP statement. Use the format under Comment Header for all scripts.
d. Scripts for Stored Procedures and User Defined Functions must include error handling.
e. Scripts that perform DML (data manipulation language) tasks must include transaction handling.
f. Always use proper indentation to increase readability.
g. Use in-line comments to describe your code, increasing readability and understandability by other programmers and DBAs.

B. Comment Header

The description for both the script and the change should be able to clearly communicate the purpose of the script to non-programmers (for example DBAs) who may have to maintain the database.

```
/****************************************************************
**    File: <filename>
**    Name: <schema>.<object>
**    Desc: <full description>
**    Auth: <developer first initial + last name>
**    Date: <date created>
****************************************************************
**    Change History
**    Date            Author         Description
****************************************************************
**    <mm/dd/yyyy>    <developer>    <description of change>
****************************************************************/
```

C. Casing

	Type of Word	Casing
a.	Keywords	uppercase
b.	Data types	lowercase
c.	Objects and Attributes	same as referenced object or attribute
d.	User defined	PascalCase (first character of each word is capitalized)

D. Error Handling

Error handling is new in SQL Server 2005. It is very similar to error handling in VB.NET. When using error handling, be sure to log your errors. It is preferable to have a table in the database to store error logging.

```
-- Variable to store ErrorLogID value of the row inserted to the log.
DECLARE @ErrorLogID int;
BEGIN TRY
    BEGIN TRANSACTION;
```

```
            <SQL statement(s)>
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        -- Call procedure to print error information.
        EXECUTE <schema>.<stored proc>;

        -- Roll back any active or un-committable transactions before
        -- inserting information in the error log table.
        IF XACT_STATE() <> 0
        BEGIN
            ROLLBACK TRANSACTION;
        END
        EXECUTE <schema>.<stored proc> @ErrorLogID = @ErrorLogID OUTPUT;
    END CATCH
    GO
```

E. Tips for Writing Good Stored Procedures

1. **Use ANSI 92**: Always try to use ANSI 92 syntax. The old syntax will work in SQL Server 2005 but will be deprecated in the next release of SQL Server. As an example, for joining use:

   ```
   SELECT <alias1>.<column>, <alias2>.<column>
   FROM <schema1>.<table1> <alias1>
       INNER JOIN <schema2>.<table2> <alias2> ON
           <alias1>.<column> = <alias2>.<column>
   ```

   Instead of:

   ```
   SELECT <alias1>.<column>, <alias2>.<column>
   FROM <schema1>.<table1> <alias1>, <schema2>.<table2> <alias2>
   WHERE <alias1>.<column> = <alias2>.<column>
   ```

2. **Minimal Use of Variables**: Use as few as possible variables. It frees space in cache.
3. **Avoid Cursors**: Avoid using cursors. Try to use temporary tables / table variables with identity column and then iterate all the tables using WHILE loop and a looping counter which will map with the identity column.
4. **Avoid the Asterisk**: Try to use only the required columns in the SELECT clause instead of using an asterisk (*). Using an asterisk returns all columns which unnecessarily create a fat recordset and may create problems when new columns are added to the underlying tables or views.
5. **Minimize Use of Dynamic Queries**: Try to minimize usage of dynamic queries. If you are using a dynamic query like `SELECT <column(s)> FROM <schema>.<table> <alias> WHERE <alias>.<column> = <@variable>` then there is no problem. You can supply a value for the <@variable> parameter and there is no recompilation of the execution plan. But if you are using a statement like `"SELECT * FROM <schema>.<table> <alias> WHERE <alias>.<column> = " + <@variable>` and supply a parameter (say 100), the cache will keep the execution plan for the value of 100 only. If the value changes (to say 101), it will recompile the

statement. Hence, this approach is slower than the previous one. TIP: You can get the exact value of the SQL statement from Profiler.

6. **Use Fully Qualified Names**: Always use the fully qualified name when calling stored procedures and functions and within stored procedures and functions. Although it is preferable to include the database name in the fully qualified name when calling, we do not use that convention because the database name would need to be altered during script promotion. Instead, begin with the schema name.
7. **Avoid SET NOCOUNT ON**: Do not use SET NOCOUNT ON whenever possible. This returns the message that shows the number of rows affected by the SQL statement. This can cause extra network traffic and can have some serious impact on performance when the procedure is called frequently.
8. **Use SET vs. SELECT**: Although a single select statement can assign values to multiple variables and is much faster than multiple SET statements assigning values to multiple variables, SET is the ANSI 92 standards compliant way to assign a value to a variable. Therefore, use the SET statement:

```
SET <@Variable1> = <value1>
SET <@Variable2> = <value2>
```

Instead of:

```
SELECT <@Variable1> = <value1>, <@Variable2> = <value2>
```

9. **Conditional Operators**: The various operators used directly affect how fast a query can run. Here are the conditional operators used in the WHERE clause, ordered by their performance: =, >, <, >=, <=, <>, !=, !>, !<.
10. **Use EXISTS vs. IN**: Try to avoid IN. While checking the existence of a set of values, use EXISTS instead of IN. IN counts the NULL values also, hence it is slower than EXISTS. Because IN returns all values and EXISTS only returns a Boolean, IN is heavier than EXISTS.

```
WHERE <alias1>.<column> NOT IN (SELECT <alias2>.<column>
     FROM <schema>.<table> <alias2>)

WHERE NOT EXISTS (SELECT <alias2>.<column>
     FROM <schema>.<table> <alias2>)
```

11. **Avoid Unnecessary Conditions**: Avoid unnecessary conditions in the WHERE clause. You can easily use the first example below instead of the second one. The second example uses an extra OR condition which can be avoided.

```
WHERE LOWER(<alias>.<column>) = 'test'

WHERE <alias>.<column> = 'TEST' OR <alias>.<column> = 'test'
```

12. **Use CAST vs**. **CONVERT**: Try to use CAST instead of CONVERT. CAST is ANSI 92 standard and CONVERT works in SQL Server only. Also, CONVERT may be deprecated in future releases of SQL Server. You may opt to use CONVERT only when you need to format the datetime datatype with the style option. CAST cannot do this.
13. **Avoid DISTINCT and ORDER BY**: If you don't need the DISTINCT or ORDER BY clauses, then do not use them. Unnecessary use of these clauses causes extra work for the database engine, making performance slower.
14. **Use UNION ALL vs. UNION**: If possible use UNION ALL instead. With UNION, any duplicates are removed, but with UNION ALL they are not. In particular, use UNION ALL if the result of each SELECT statement is distinct. NOTE: UNION and UNION ALL do not guarantee the order in the result.
15. **Use Table Variables vs. Temporary Tables**: Temporary tables can cause stored procedures to recompile. But table variables were designed specifically to guard against stored procedure recompiles during execution.
16. **Use Indexes Properly**: The data tuning advisor can help identify where to use indexes, but it does not give the proper result every time. Index scans are much faster than table scans. So, identify the table scans from the execution plan. But when a table returns smaller rows, it is better to use a table scan.
17. **Use Profiler to Identify Cache Usage**: The CacheMiss event class indicates that the stored procedure is not in the cache. If the SP:CacheMiss class occurs frequently, it can indicate that more memory should be available to SQL Server, thereby increasing the size of procedure cache.
18. **Avoid NOLOCK**: Starting with SQL Server 2005 with the introduction of the snapshot isolation, there is no justification whatsoever to use NOLOCK. Do not use and hints (NOLOCK is a hint) until your testing proves that you have an issue that can't be solved any other way. If you use a hint, prove it via testing and document it.