Dartec Systems

Data Quality Engine (DQE)

Technical Overview

Table of Contents

1 Overview

DQE is a set of routines designed to run on the SQL Server platform. A user guide that describes operating and monitoring activities can be found in the User Guide document.

The DQE is designed as a SSIS, MDS, SQL Agent and T-SQL based solution. The DQE is capable of profiling, validating and cleansing data based on pre-defined DQ Rules.

The DQE can be used at any stage in the ETL process and can be used to implement multi-step activities.

This document covers the Technologies used, the different type of rule, the structure of the application, the data model and an overview of the BI reports

2 Technologies

The DQE solution has been delivered using a number of technologies.

Technology	**DQE Components**
Windows 2012	Operating system platform
SQL Server 2012	Database (DataQualityDB) SQL Scheduling Agent (DataQualityEngineJob)
SQL Server 2012 SSIS	Data Quality Engine Project (DataQualityEngine)
SQL Server 2012 SSRS	Reporting Services Project (DQEReports)
SQL Server 2012 MDS	MDS Model (DataQuality)
Visual Studio 2012	Development environment

3 DQE (Data Quality Engine) Design

DQE is based on an open-source SQL Server based data profiling and cleansing tool made publically available by Dartec Systems. The DQE is free to alter and configure based on specific and local requirements. Dartec Systems do not provide support or maintenance for the DQE.

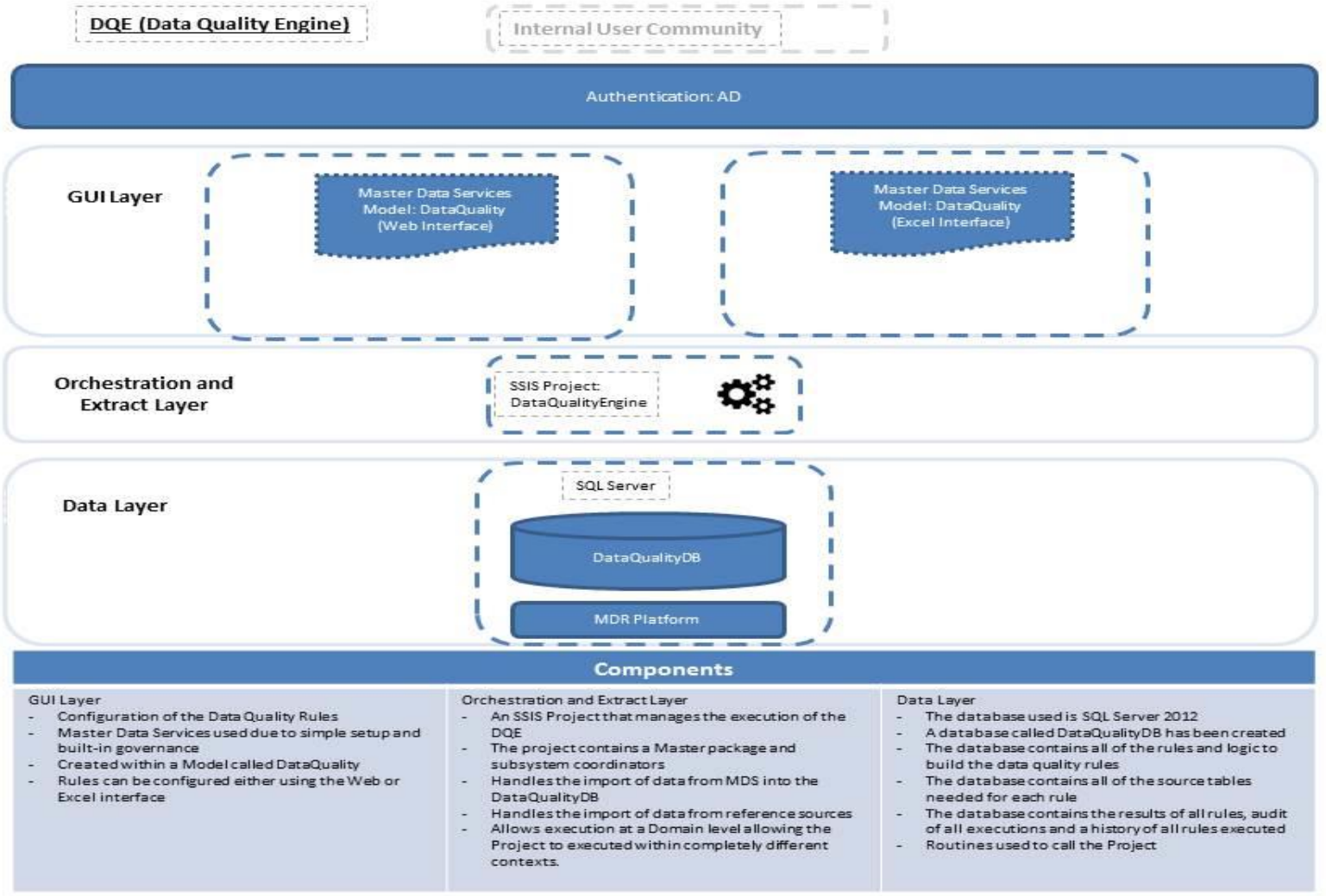
DQE has been designed to

- Centralise data quality rules into a single repository
- Provide a user interface to allow technically-oriented users to manage rules
- Provide a flexible mechanism for invoking the data quality rules such that it can be easily be called at multiple points within an ETL flow
- Capture the results of past tests and keep a record of failed DQ tests

DQE has been built using the Microsoft SQL Server Stack which consists of

- Master Data Services to provide an UI either via the Web Page or within Excel
- SSIS to orchestrate the extraction and data quality activities
- T-SQL to dynamically build, execute and capture the data quality rules based on the configurations found within the Master Data Services model.

The architecture of DQE is illustrated in the below diagram and each layer of the application is described further below.



Components

GUI Layer
- Configuration of the Data Quality Rules
- Master Data Services used due to simple setup and built-in governance
- Created within a Model called DataQuality
- Rules can be configured either using the Web or Excel interface

Orchestration and Extract Layer
- An SSIS Project that manages the execution of the DQE
- The project contains a Master package and subsystem coordinators
- Handles the import of data from MDS into the DataQualityDB
- Handles the import of data from reference sources
- Allows execution at a Domain level allowing the Project to executed within completely different contexts.

Data Layer
- The database used is SQL Server 2012
- A database called DataQualityDB has been created
- The database contains all of the rules and logic to build the data quality rules
- The database contains all of the source tables needed for each rule
- The database contains the results of all rules, audit of all executions and a history of all rules executed
- Routines used to call the Project

The **GUI Layer** is the user interface where users would setup and manage data quality rules:

- The GUI allows the configuration of the Data Quality Rules, allowing new rules to be created, updated and deleted;
- Master Data Services component of SQL Server has been used as it contains a large degree of built-in auditing and governance allowing rule management to be secured and change tracked;
- A stand-alone MDS Model has been created named “DataQuality”
- MDS Rules can be configured either using the Web or Excel interface

The **Orchestration and Extract Layer** has been implemented using SSIS. Generally SSIS has been employed in a relatively simple way and the complexity is pushed to the database level:

- A SSIS Project named “DataQualityEngine” has been created
- The SSIS project contains one master orchestration package and three child subsystems
    - 1. Import data from the MDS database subscription views
    - 2. Import data from reference sources
    - 3. Coordinate the execution of the data quality rules
- Individual packages have been created to import MDS subscription views into the DataQualityDB
- Individual packages will need to be created to import of data from reference sources
- The SSIS project is stored in the SSIS catalog and is best called using a Stored Procedure contained in the DataQualityDB “DQ.sExecuteJobDataQualityEngine”
- The above Stored Procedure is used to set runtime parameters and handles the execution of a SQL Server scheduled group “DataQualityEngineJob”
- Based on the runtime parameters the DQE will execute a specific set of data quality rules (aka DQ Rules) which is referred to, through-out, as a DQ Domain of rules.

The majority of DQE's logic and auditing is contained in the **Data Layer**. The Data Layer has been implemented using a SQL Server database

- The database name is "DataQualityDB"
- The database contains all of the rules and logic to build the data quality rules
- The database contains a MDS and Reference schema to hold data imported from MDS or as a reference table.
- The database contains tables and views to help report the results of all rules, auditing to report on all executions and a history of all rules syntax executed for trouble-shooting purposes.
- The database also contains some basic utility routines to help execute the SSIS Project using T-SQL.

The Data Quality layer contains the logic and code needed to supports a number of different types of data quality categories and rule types detailed (together with outputs) in the next section.

The below table illustrates the categories and types of rule that the DQE has been designed to accommodate together with the types of outputs each rule will generate.

3.1 Data Quality Rules and Behaviours

DQE contains a number of different rule categories and each contains a number of rule types. Each rule behaves slightly differently and produces different types of output. In some cases the outputs simply keep a record of the number successful or failure DQ tests. In other cases, the rules will either update the column being evaluated or will write the output to a specific output or status column.

The below tables summarises these behaviours. These rule behaviours may still require some tidying up and tweaking to get functionality needed.

Rules Information			**Outputs**				
Rule Category	**Rule Types**	**Description**	**Output Column**	**Status Column**	**Affected Rows**		**Test Results Summary**
					DQ Ids (DQ.DataQuality RowHistory)	Primary Keys (DQ.DataQuality PrimaryKeyValues)	**(DQ.DataQuality History)** **&** [Reports].[DQSummaryResults]
Expression Rules	**Default**	Example: RuleAssociationCode **(21)** Evaluates the expression, inserts the records that satisfy the expression into the DataQualityRowsHistory table, inserts the primary key of the records that satisfy the			**Yes** Records the DQ Id of the records that satisfy the expression	**Yes** Records the primary key of the records that satisfy the expression.	**Yes** Records a count of the number rows matching the expression rule.

		expression into the DataQualityPrimaryKeyValues table and inserts a count of the number of rows matching the expression into the DataQualityHistory table.					
	Indicator	Example: RuleAssociationCode **(41)** As with 'Default' rule type plus:- **creates** an indicator flag on the DQEntity.	If a value is specified for the "OutputColumn" a new column, named after this value, is created and used. Column is populated with a '1' to indicate that a match to the expression has been found on that row (or 'populated with a 'NULL' where no match is found)	**Optional:** Status plays no role here and does not need to be defined	**Yes** Records the DQ Id of the records that satisfy the expression	**Yes** Records the primary key of the records that satisfy the expression.	**Yes** Records a count of the number of records matching the expression rule.
	Delete	Example: RuleAssociationCode **(27 & 209)** As with 'Default' rule type plus:- **deletes** the records that			**Yes** Records the DQ Id of the records that satisfy the expression	**Yes** Records the primary key of the records that satisfy the expression.	**Yes** Records a count of the number of records matching the expression rule

		match the entity from the DQEntity table. NOTE: There are two ways of using this. 1. Enter the full expression into the Expression column. This requires the use of the IGNORE keyword. Eg. 27 2. Use the evaluation column. E.g 209					
Harmonization Rules	**ToLower**	Example: RuleAssociationCode **(22)** Sets all characters to lower case in the specified Evaluation Column.	**Optional:** If a value is specified for the 'OutputColumn, then the harmonized values are written to this Output column (column is named as specified by value in "OutputColumn". If no output column value is explicitly specified then the	**Optional:** If a value is specified for the "StatusColumn", then a new column of this name is created and used. If no value is specified in the "StatusColumn", then , the system creates a status column using a default column name named as per the	**No**	**No**	**Yes** Records a count of the number of records harmonized

			data in the EvaluationColumn itself will be harmonized. This is true of all rules within the Harmonization Category.	RuleEntity_Association Code being run e.g. 'StatusColRule_130. Values within the Status column are:- '<RuleName>: Applied' - where a harmonisation has been made or 'Null' where harmonisation has been applied.			
	ToUpper	Sets all characters to UPPER case in the specified Evaluation Column.	**Optional:** (See "ToLower" rule for full description)	**Optional:** (See "ToLower" rule for full description)		**Yes**	**Yes** Records a count of the records set to Upper count.
	Remove Spaces	Strips all spaces from the EvaluationColumn's values.	**Optional:** (See "ToLower" rule for full description)	**Optional:** (See "ToLower" rule for full description)		**Yes**	**Yes** Records a count of the records updated.
	Remove Specified Character	Example: RuleAssociationCode **(210)** Strips the specified value from the EvaluationColumn value.	**Optional:** (See "ToLower" rule for full description)	**Optional:** (See "ToLower" rule for full description)		**Yes**	**Yes** Records a count of the records updated.

	Set Blanks As Null	Example: RuleAssociationCode **(30)** Finds any blank ‘’ values within the EvaluationColumn and explicitly sets the value to NULL.	**Optional:** (See “ToLower” rule for full description)	**Optional:** (See “ToLower” rule for full description)		**Yes**	**Yes** Records a count of the records updated.
	Set Null As Default Value (String)	Example: RuleAssociationCode **(31)** Finds any NULL values within the EvaluationColumn and explicitly sets the cell to the value provided in the ReplacingValue column.	**Optional:** (See “ToLower” rule for full description)	**Optional:** (See “ToLower” rule for full description)		**Yes**	**Yes** Records a count of the records updated.
	Set Null As Default Value (Date)	Example: RuleAssociationCode **(32)**	**Optional:** (See “ToLower” rule for full description)	**Optional:** (See “ToLower” rule for full description)		**Yes**	**Yes** Records a count of the records updated.
	Special Operation	Embeds the bespoke function specified in the “BespokeFunction” column which is applied to the EvaluationColumn. e.g. The function ‘ReturnNumericAlphaMask’ could return an alpha numeric	**Optional:** (See “ToLower” rule for full description)	**Optional:** (See “ToLower” rule for full description)		**Yes**	**Yes** Records a count of the records updated.

		mask of a selected evaluation column (e.g. 'NNA-NNA')					
	Replace Value	Example: RuleAssociationCode **(211)** Replaces the pattern specified in the "SpecifiedCharacter" column with the pattern found in the "ReplacingValue" column.	**Optional:** (See "ToLower" rule for full description)	**Optional:** (See "ToLower" rule for full description)		**Yes**	**Yes** Records a count of the records updated.
	Confirm UK Date Format (dd mm yyyy)	Confirms that the varchar is a valid UK date of the format dd/mm/yyyy i.e. 01/12/2016 is valid whereas 01/13/2016 is invalid. Also checks leap year dates are valid. Typically this will be done before a transformation to date is attempted. The output of this check can be used as a condition in the transform rules.	Outputs 1 to signify the varchar is a valid UK date	Simply indicates that the rule has been applied.		**Yes**	**Yes** Records a count of the records updated.
	Confirm US Date Format (mm dd yyyy)	Confirms that the varchar is a valid US date of the format mm/dd/yyyy i.e. 12/24/2016 is valid whereas 13/24/2016 is	Outputs 1 to signify the varchar is a valid US date	Simply indicates that the rule has been applied.		**Yes**	**Yes** Records a count of the records updated.

		invalid. Also checks leap year dates are valid.					
	Get Alpha Numeric Mask Pattern	This rule can be used to collect information about the format and ranges of values supplied within a given field. This rule will identify the structure of a value in terms of whether it is an Alpha (A) or Numeric (N). For example, 22-AD21 would be presented as NN-AANN.	Outputs the pattern mask	Indicates that the rule has been applied			**Yes** Records a count of the records checked
Value Correction Rules	**Value Correction**	**Standard Example:** RuleAssociationCode **(29)** Updates the Evaluation Column so that the SourceValues is replaced with the value specified in the PreferredValue column. **Ruleset Example:** RuleAssociationCode **(18)** It is a common scenario that multiple values may need to be corrected and as such rulesets are be used to	**Optional:** If specified, the corrected values are written to the Output column specified in "OutputColumn". If no OutputColumn is specified explicitly, then the corrected values output overwrites the original data in the	**Optional:** If a value is specified for the "StatusColumn", then a new column of this name is created and used. If no value is specified in the "StatusColumn", then , the system creates a status column using a default column name named as per the	**Yes** The DQId of each and every corrected record.	**Yes** The primary keys of each and every corrected record. .	**Yes** Summary that records:- Number of records that were corrected,Number of records already correct, and Number of records where no correction rules could be applied e.g. values = 'null'.

		improve the ValueCorrection rules. A ruleset simply groups a number of lower level rules.	"EvaluationColumn."	RuleEntity_Association Code being run e.g. 'StatusColRule_130. Values within the Status column are:- 'Corrected'- where a value correction has been made. 'Correct' - where a value correction does not need to be made Warning: No Rule Applied' – where a correction cannot be made e.g. value is Null			
Reference Rules	**Table Reference**	**Example:** RuleAssociationCode **(19)** Compares the EvaluationColumn against the values in an external reference table. Because the external reference table must be available to a Stored Procedure located in the DataQualityDB, an SSIS package (created within the	**NA**	This rule will only create one column. If a value is specified for the "StatusColumn" OR the Output column then a new column is created and used. If an output and status are defined, the status column value will be	**Yes** The DQId of each record failing the integrity check.	**Yes** The primary key of each and every record failing the integrity check..	**Yes** Summary information about the number of records failing the integrity check.

		ImportReference.Controller subsystem) may be required to import the reference table as run time.		used and the output column value ignored. If no value is specified in the “StatusColumn” a default status column will be created and updated (named as per the RuleEntity_Association Code being run e.g. ‘StatusColRule_130). Column is populated with same text as OutputColumn depending on the result of the integrity check.			
	List Reference	**Example:** RuleAssociationCode **(38)** Compares the EvaluationColumn against the values in an AppReferenceLists table. A new reference list can be created within the MDS model and is ideal for small controlled lists.	NA	This rule will only create one column. If a value is specified for the “StatusColumn” OR the Output column then a new column is created and used. If an output and status are defined, the status column value will be	**Yes** The DQId of each record failing the integrity check.	**Yes** The primary key of each and every record failing the integrity check.	**Yes** Summary information about the number of records failing the integrity check.

				used and the output column value ignored. If no value is specified in the “StatusColumn” a default status column will be created and updated (named as per the RuleEntity_Association Code being run e.g. ‘StatusColRule_130). Column is populated with same text as OutputColumn depending on the result of the integrity check.			
	Referential Integrity	**Example:** RuleAssociationCode **(212)** This rule joins the DQEntity and the Reference Entity and attempts to establish whether any records exist in the DQEntity that do not exist in the Reference Entity. Where this is found it is regarded as an integrity issue.	NA	This rule will only create one column. If a value is specified for the “StatusColumn” OR the Output column then a new column is created and used. If an output and status are defined, the status column value will be	**Yes**	**Yes**	**Yes** Summary information about the number of records failing the integrity check.

		The rule uses the JoinLogic column in the RuleReference entity to apply a LEFT OUTER JOIN between the Reference Database/ Schema and Column (the B alias table on the LEFT) and the DQEntity (the A alias table on the right). The DQEntity is defined in the RuleEntity Association table. The AttributesComparisons field is used to state the primary key on the B. alias side that you should evaluate as NULL when a LEFT OUTER JOIN is applied.		used and the output column value ignored. If no value is specified in the "StatusColumn" a default status column will be created and updated (named as per the RuleEntity_Association Code being run e.g. 'StatusColRule_130). Column is populated with same text as OutputColumn depending on the result of the integrity check.			
	Attribute Comparisons	**Example:** RuleAssociationCode **(208)** Uses the information provided in the RuleReference table to join two tables and look for differences in attributes. This rule type required the ON portion of the JOIN to be manually coded and the		If a value is specified for the "StatusColumn" (and no value is specified for the OutputColumn) then a new column is created and used. If no value is specified in the "StatusColumn" a default status column will be created and	**Yes** The DQId of each record failing the integrity check.	**Yes** The primary key of each and every record failing the integrity check.	**Yes** Summary information about the number of records failing the integrity check.

		attribute not equals to be manually coded.		updated (named as per the RuleEntity_Association Code being run e.g. 'StatusColRule_130). Column is populated with "FAILURE: Join established but attribute comparison failed" if no join is found otherwise it is populated with "SUCCESS: Join established and attribute matched"			
Profiling Rules	**Data Type Check**	**Example:** RuleAssociationCode **(213)** Attempts to CAST the EvaluationColumn into the specified datatype. This rule will check whether the values stated within the Evaluation column can be cast into the data type sepcified in the RuleProfiling	**NA**	**Optional:** If specified in the "StatusColumn" it will be populated with a binary flag WHERE 1 signifies an exception and 0 signifies no issues encountered.	**Yes** The DQId of each failed record.	**Yes** The primary keys of corrected records will be retained.	**Yes** Inserts a count of failed records.

		rule (which must be of a ProfileType 'DataTypeCheck'). The outputs will be written to the status column if defined otherwise the results are simply written to the results tables.					
	Duplicates Count	Based on the PrimaryKey specified on the DQEntity this type identifies duplicates in the DQEntity. The outputs are logged simply logged but not flagged.			**Yes** The DQId of each duplicated record.	**Yes** The primary keys of corrected records will be retained.	**Yes** Inserts a count of duplicate records.
	Duplicates Flag	**Example:** RuleAssociationCode **(23)** Based on the PrimaryKey specified on the DQEntity this type identifies duplicates in the DQEntity. The outputs are logged and a mandatory status column is written to.		**Mandatory:** If specified in the "StatusColumn" the rule will flag all duplicate records but distinguish between the 'first' duplicate (indicated by a value of '1') and the other duplicates (indicated by a value of '2').	**Yes** The DQId of each duplicated record.	**Yes** The primary keys of each duplicated record..	**Yes** Inserts a count of the total number of duplicate records and also a count of the number of duplicated primary keys.

	Min And Max Value Profile	**Example:** RuleAssociationCode **(36)** This rule will log the Min and Max values found within the DQEntity. If the EvaluationColumn value equals ALL, Then rule will record the Min and Max values for all columns within the DQEntity. If the EvaluationColumn specifies a single column name, the min and max values are only captured for that single column.	**NA**	**NA**	NA	NA	**Yes** Inserts the Min and Max values for the column or columns specified.
	Min And Max Length Profile	**Example:** RuleAssociationCode **(37)** This rule will log the Min and Max value lengths found within the DQEntity. If the EvaluationColumn value equals ALL, Then rule will record the Min and Max value length for all columns within the DQEntity.	**NA**	**NA**	NA	NA	**Yes** Inserts the Min and Max value lengths for the column or columns specified.

		If the EvaluationColumn specifies a single column name, the min and max value length are only captured for that single column.					
	Table Value Distribution Profile	**Example:** RuleAssociationCode **(33)** The Evaluation column should be 'ALL'. This rule will evaluate all columns within the table. This rule profiles a DQEntity, assessing each column within the DQEntity to establish the following - Row Count - Count of populated rows - Count of empty rows (NULL or '') - Count of duplicates within each column - Count of distinct rows	**NA**	**NA**	NA	NA	**Yes** Profile the entire DQEntity and capture the information in the 'Description' column.

		The results are logged to the DataQualityHistory table					
	Column Value Distribution Profile	**Example:** RuleAssociationCode **(35)** This rule will gather the value distribution for the column specified in the Evaluation column. The output is a percentage and count breakdown of all values with a distribution above the specified threshold. The threshold value found in the Threshold column of the RuleProfile table. Any values that make up the threshold value or less will be grouped together into the category of 'MiscMinorValues'. Note, if you want to see the distribution of all values set the threshold to 0.					**Yes** Capture the counts and percentages of the value distribution in the column specified.
	Table Count	**Example:** RuleAssociationCode **(34)**	**NA**	**NA**	NA	NA	**Yes** Capture the row count of the DQEntity.

		Returns the row count of a table. The result is stored to the audit and results tables. The evaluation column should be 'IGNORE' for this type of rule.					
Transformation Rules	**Varchar to Integer**	Converts a data type of varchar to a data type of integer (usually to allow subsequent rules to be applied to the integer type).	Yes			**Yes**	**Yes**
	Integer to Varchar	Converts an Int to a Varchar	Yes			**Yes**	**Yes**
	Varchar(UK) to Date/Time	Converts a data type of varchar(UK) to a data type of DateTime (usually to allow subsequent rules to be applied to the new data type)	Yes			**Yes**	**Yes**
	Varchar(US) to Date/Time	Converts a data type of varchar(US) to a data type of DateTime (usually to allow subsequent rules to be applied to the new data type)	Yes			**Yes**	**Yes**

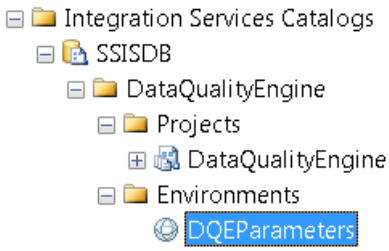
	Varchar(UK) to Integer Date/Time	Converts a data type of varchar(UK) to a data type of IntegerDateTime (usually to allow subsequent rules to be applied to the new data type) e.g. '13/01/2016' is converted to '20160113'	Yes			**Yes**	**Yes**
	Varchar(US) to Integer Date/Time	Converts a data type of varchar(US) to a data type of IntegerDateTime (usually to allow subsequent rules to be applied to the new data type) e.g. '13/01/2016' is converted to '20160113'	Yes			**Yes**	**Yes**

4 The SSIS Project: DataQualityEngine

The SSIS Project is a fundamental component of the DQE application and is responsible for orchestrating each step of the DQE processes and applying either DQ Rules or DQ Domains.

4.1 SSIS Storage

The SSIS Project is stored within the SSIS Catalog that contains one Project and one set of environment parameters.



The project is named 'DataQualityEngine' and contains an environments file is named 'DQEParameters'. *The Project Parameters held in this environments file are critical in controlling how the DQE executes.*

The parameters stored are

Parameter	*Description*
ParentLoadId	The LoadId of the routine that calls the SSIS project. When DQE is executed using the provided Stored Procedures this value is updated at runtime.
DomainName	This parameter is used to control the domain of rules executed by the DQE. When DQE is executed using the provided Stored Procedures this value is updated at runtime.
Rule Entity Association Code	Optional. This value can be used to execute a single rule rather than a DQ Domain of multiple rules. When DQE is executed using the provided Stored Procedures this value is updated at runtime.

MDSDB_ConnectionString	The connection string to the MDS database that holds the subscription views of DQE Configuration tables
DataQualityDB_ConnectionString	The connection string to the DataQualityDB database that contains all of the DQE stored procedures and tables
AuditDB_ConnectionString	The connection string of the database that contains all of the DQE auditing tables. By default this is same as the DataQualityDB.

4.2 SSIS Project Overview

DQE follows a modular design in terms of how the DQE is invoked and also in terms of how the application is written.

The DQE orchestration process is coordinated using an SSIS project. This project is responsible for managing the sequence of activities handled by DQE. This approach is key in creating a predictable, repeatable and automated execution mechanism.

The subsystems of the DQE orchestration process are

- DQE Orchestration Controller (**MasterController.dtsx**)
- Import MDS Data (**ImportMDS.Controller.dtsx)**
- Import Reference Data (**ImportReference.Controller.dtsx**)
- Data Quality Engine (**DataQualityEngine.dtsx)**

Generally, all packages follow the same simple template. Each package contains the following common elements

Control Flow Tasks:

- Audit Start: Writes start time to DataQualityDB.Audit.RoutineLoad & generates a loaded associated to that routine.
- Audit End: Writes end time to DataQualityDB.Audit.RoutineLoad & generates a loaded associated to that routine.
- OnFailure: Audit Fail: Writes failure time to DataQualityDB.Audit. RoutineLoad & generates a loaded associated to that routine.

- OnFailure: Log error message: Writes error details to DataQualityDB.Audit. RoutineError & generates a loaded associated to that routine.
- Sequence Container for application logic

All auditing is linked through the LoadId generated by the Audit Start call.

The subsystems of the DQE are explained in more detail below

4.2.1 Component: DQE Orchestration Controller

SSIS Path: MasterController.dtsx

Subsystem controller: ImportMDS.Controller.dtsx

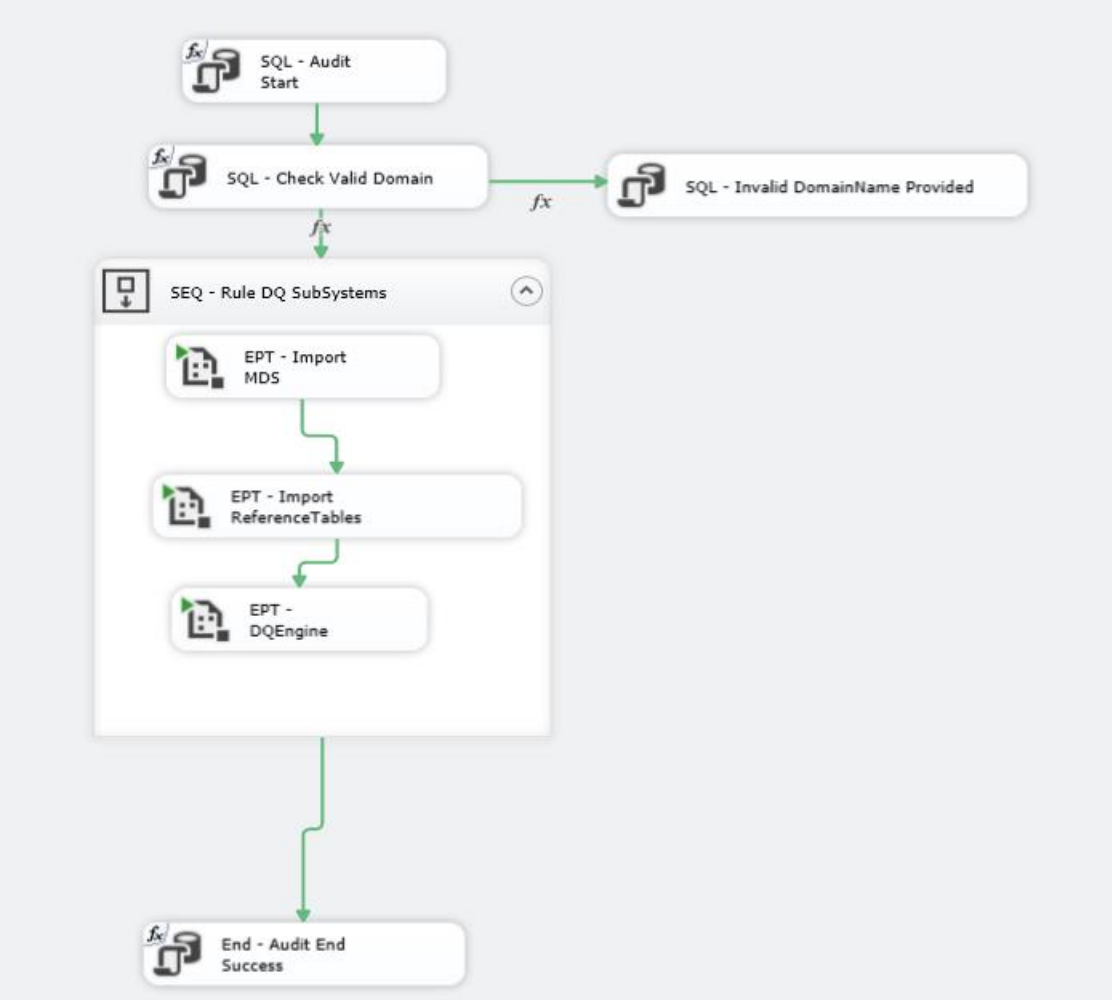
Action mechanism: N/a

This package is primarily responsible for executing each component in sequence.

The MasterController is designed to be executed with a number of parameters handed into the Project. These include

- Parent LoadId
- DQ Domain to Execute
- DQ Rule to run (Optional if a StandAlone Rule is being executed)
- Connection Strings (AuditDB; DataQualityDB; MDS DB)

The Master Controller will first evaluate the provided Domain Name to ensure that it corresponds to an existing Domain. If the Domain name is evaluated successfully, the Master controller will execute each of the below subsystems in sequence. When the subsystem package are called the LoadId generated by the MasterController is handed down as the ParentLoadID (this parent-child information is retained in the auditing tables).



Where required, each subsystem will contain its own coordinating mechanism, which will be responsible for executing the leaf level activities.

4.2.2 Component: Import MDS Subsystem

SSIS Path: MasterController\ ImportMDS.Controller.dtsx\ Import package per table

Subsystem controller: ImportMDS.Controller.dtsx

Action mechanism: Package per table import

Loads the DQE configuration data entered via the MDS front-end into the DataQualityDB. Once loaded this configuration data is used to apply the DQ Rules within the domain specified by the DomainName parameter.

Each action package is called by the controlling package. When the action packages are called the LoadId generated by the ImportMDS.Controller.dtsx is handed down as the ParentLoadID (this parent-child information is retained in the auditing tables).

Each package will load a table from the MDS database into the MDS schema in the DataQualityDB.

Each package follows an identical pattern of truncate the destination table and fully import the data from the MDS Subscription views.

4.2.3 Component: Import Reference Tables Subsystem

SSIS Path: MasterController\ ImportReference.Controller.dtsx \ Import package per table

Subsystem controller: ImportReference.Controller.dtsx

Action mechanism: Package per table import

This subsystem is included to import data from external sources so that DQ Rules can use the data - for example when running External Reference Rules.

Each action package is called by the controlling package. When the action packages are called the LoadId generated by the ImportReference.Controller.dtsx is handed down as the ParentLoadID (this parent-child information is retained in the auditing tables).

These packages can be used to load data to location on the same server that the DataQualityDB is located on.

Each package follows an identical pattern of truncate the destination table and fully import the data from the reference data source.

4.2.4 Component: Data Quality Engine

SSIS Path: MasterController\ DataQualityEngine.dtsx

Subsystem controller: DataQualityEngine.dtsx

Action mechanism: DataQualityEngine.dtsx

The data quality rules are applied within a single, relatively complex package (i.e. relative to the other action packages within the Project).

The package makes an initial decision regarding whether to execute a stand-alone rule or execute all rules within the provided DQDomain.

4.2.4.1 Stand-Alone Rule

This decision is based on the project parameter “RuleEntityAssociationCode”. If the default ‘ignore’ value is overwritten with a RuleEntityAssociationCode the

[DQ].[sExecuteStandAloneRule] is called and the RuleEntityAssociationCode value is provided as a parameter input. A single rule is executed and all other logic within the package is subsequently by-passed.

4.2.4.2 Domain Rule Execution

If the package is confirmed as **not** a stand-alone rule execution, the package continues along the standard path.
This starts with the execution of DataQualityDB.`[DQ].[sGetDomainEntities]`.
This procedure serves a dual purpose

- Creates 'cleansing' versions of the tables if specified in the metadata
- Gets the details of all active rules within the specified Domain (driven by the **DomainName** project parameter).

The project establishes what execution sequences are needed within this Domain. For example, if three rules are defined on execution sequence 1,2,3 respectively, this step will identify that 3 execution iterations are required. This count is fed into an enumeration container (which will execute once for each iteration of the execution sequence).

For each execution sequence a list of rules are picked up and fed into an enumerator, which executes once for each rule.

For each rule, the rule type is evaluated and the appropriate stored procedure is executed and the RuleEntityAssociationCode is provided as an input.

There are five categories of rule

- `[DQ].[sApplyDQRuleExpression]`
- `[DQ].[sApplyDQRuleHarmonization]`
- `[DQ].[sApplyDQRuleProfiling]`
- `[DQ].[sApplyDQRuleReferences]`
- `[DQ].[sApplyDQRuleValueCorrect]`

If any of the rule executions fail the process will log the failure and continue with the other queued rules.

Once all active rules within a domain are applied the process completes and closes the job.

5 The Data Layer: DataQualityDB

The DataQualityDB is **the** fundamental component of the DQE application and is responsible for building the data quality rules, applying the rules to the data, logging the test results, auditing all rule execution activities and keeping a record of all rules applied to the data.

5.1 Data Model

The DQE contains two subject areas of data

- Rule Configuration
- Results and Activity Logging

Each subject area contains a set of tables relating to a specific set of activities. Foreign key relationships ensure that all relevant tables can be linked. Each subject area is explained below.

5.1.1 The Rule Configuration Subject Area aka the GUI Data Model

The Rule configuration subject area is split into three groups of tables.

- Domain, Entity and Application Configuration Tables
- Rule Configuration Tables
- Entity_Rule Linkage Table

The “Domain, Entity and Application Configuration” and “Rule Configuration Tables” are linked through the “Entity_Rule Linkage Table”.

The “Domain, Entity and Application Configuration” tables are used to create new DQ Domains and configure the entities (i.e. database tables) that will have rules applied to them.

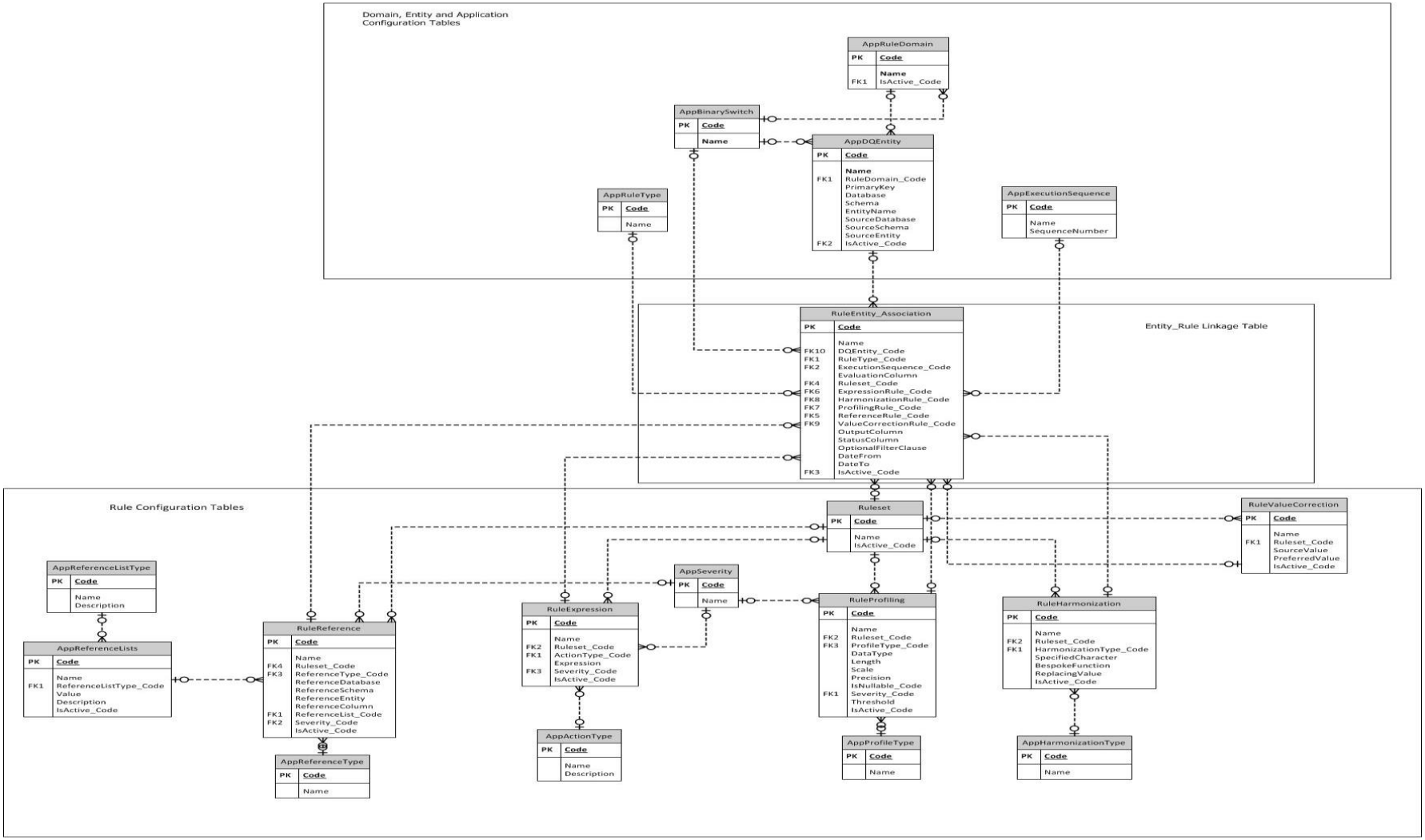
The “Rule Configuration Tables” are used to create new and configure existing rules. A rule can be defined once and using the linkage table applied to many entities.

The Linkage table is used to apply Data Quality Rules to Entities (i.e. Tables). This allows individual, generic rules to be applied many times.

The data model is partly constrained by and reflects the functionality available within Master Data Services (i.e. relatively de-normalised to aide usability).

The Data model is presented through an MDS model “DataQuality”, **this forms the foundation of the GUI**. These tables will be accessible via the default MDS web interface and the excel plug-in. This data will also be made available via pre-created MDS subscription views. Because it is assumed that the Data Quality database will reside on a different server, to MDS, the SSIS project extracts data from these views as part of the load process and loads the data into a set of DataQualityDB located staging tables.

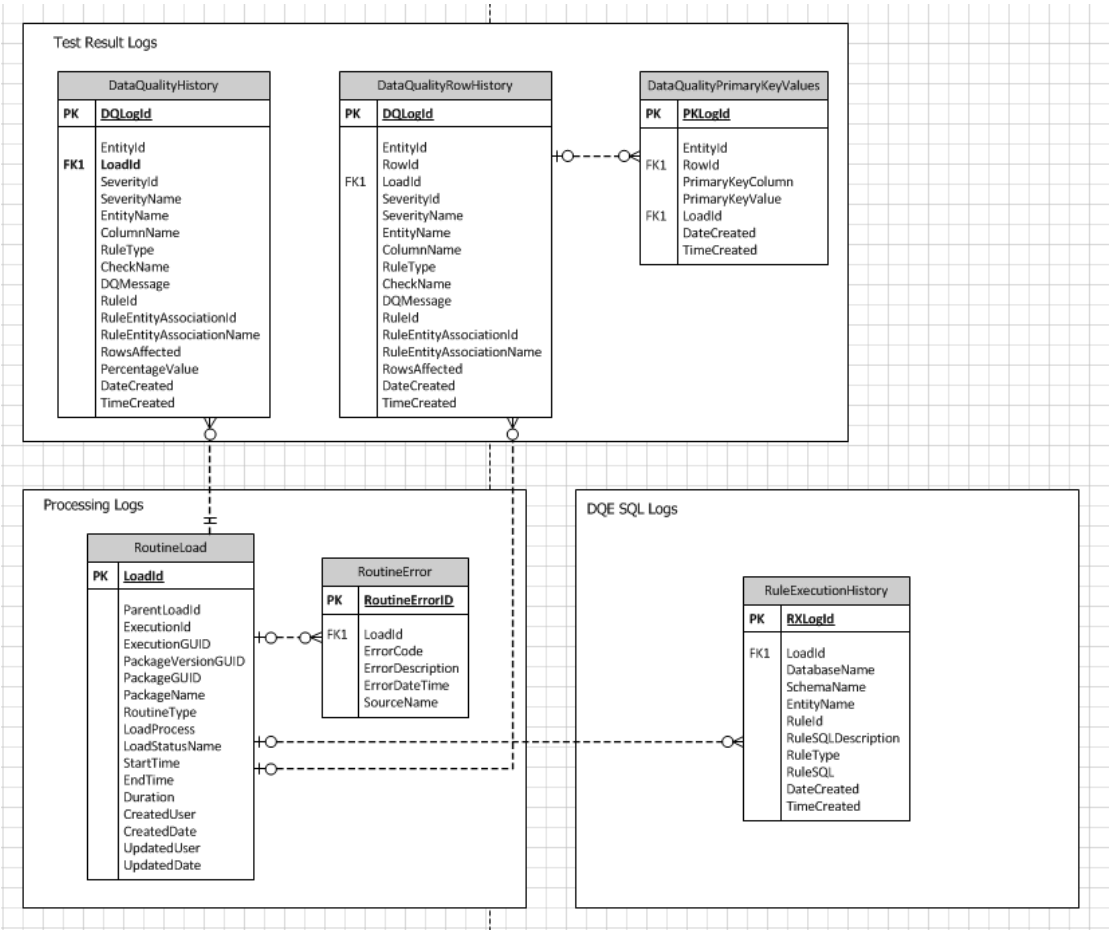
The below diagram illustrates the Rule Configuration Subject Area.


Domain, Entity and Application Configuration Tables
AppRuleDomain
PK Code
Name
FK1 IsActive_Code
AppBinarySwitch
PK Code
Name
AppDQEntity
PK Code
Name
FK1 RuleDomain_Code
PrimaryKey
Database
Schema
EntityName
SourceDatabase
SourceSchema
SourceEntity
FK2 IsActive_Code
AppRuleType
PK Code
Name
AppExecutionSequence
PK Code
Name
SequenceNumber
RuleEntity_Association
PK Code
Name
FK10 DQEntity_Code
FK1 RuleType_Code
FK2 ExecutionSequence_Code
EvaluationColumn
FK4 Ruleset_Code
FK6 ExpressionRule_Code
FK8 HarmonizationRule_Code
FK7 ProfilingRule_Code
FK5 ReferenceRule_Code
FK9 ValueCorrectionRule_Code
OutputColumn
StatusColumn
OptionalFilterClause
DateFrom
DateTo
FK3 IsActive_Code
Entity_Rule Linkage Table
Rule Configuration Tables
Ruleset
PK Code
Name
IsActive_Code
RuleValueCorrection
PK Code
FK1 Name
Ruleset_Code
SourceValue
PreferredValue
IsActive_Code
AppReferenceListType
PK Code
Name
Description
AppReferenceLists
PK Code
FK1 Name
ReferenceListType_Code
Value
Description
IsActive_Code
RuleReference
PK Code
Name
FK4 Ruleset_Code
FK3 ReferenceType_Code
ReferenceDatabase
ReferenceSchema
ReferenceEntity
ReferenceColumn
ReferenceList_Code
FK1 Severity_Code
FK2 IsActive_Code
AppReferenceType
PK Code
Name
AppSeverity
PK Code
Name
RuleExpression
PK Code
Name
FK2 Ruleset_Code
FK1 ActionType_Code
Expression
FK3 Severity_Code
IsActive_Code
AppActionType
PK Code
Name
Description
RuleProfiling
PK Code
Name
FK2 Ruleset_Code
FK3 ProfileType_Code
DataType
Length
Scale
Precision
IsNullable_Code
FK1 Severity_Code
Threshold
IsActive_Code
AppProfileType
PK Code
Name
RuleHarmonization
PK Code
Name
FK2 Ruleset_Code
FK1 HarmonizationType_Code
SpecifiedCharacter
BespokeFunction
ReplacingValue
IsActive_Code
AppHarmonizationType
PK Code
Name

5.1.2 Results and Activity Logging Subject Area

The Results and Activity Logging is split into three groups of tables

- Data Processing logs: Each step within the DQE code is recorded together with a distinct LoadId, Name, Stored Procedure or SSIS package name, Start time, end time and the success status
- DQE Test Result logs: Each DQ Rule generates a test output together with descriptions, test results and the number of records identified
- DQE SQL logs: Most activities within the DQE are based on dynamic SQL. Each 'chunk' of actual SQL produced is logged.



5.2 Data Quality DB Entity Descriptions

The above tables are created on specifically defined database schemas. Each database object is listed and explained below.

Schema: Audit	**This schema holds tables, views and stored procedures relating to the auditing of ETL activities.**	
Table	tRoutineload	Each time a routine (stored procedure or SSIS package) is started and stopped a record is written to this table.
Table	tRoutineError	Each time an error is encountered error information is written to this table with the associated LoadId.
View	EntitySizes	A utility table that provides information about the tables within the database.
View	RoutineLoadHistory	This view presents a user friendly presentation of the ETL load audit data.
Stored Procedure	sStartRoutineLoad	A utility routine that logs the start of a routine and generates a unique LoadId for that routine. All actions associated to that routine (or its children) can be tidied back to this LoadId.
Stored Procedure	sEndRoutineLoad	A utility routine that stamps the end time and success status onto each LoadId.
Stored Procedure	sRoutineErrorStamp	A utility routine that collects and logs error information.
Schema: MDS	**This schema holds the configuration tables imported from the MDS DataQuality model. *These entities have been described in An earlier section***	

Table	AppRuleDomain	Contains the DQ Domains that will be used to group DQ Rules at execution time.
Table	AppDQEntity	This table is used to define the DQEntities (tables) that data quality rules can be applied to
Table	RuleEntity_Association	This table is used to link DQ Rules to DQEntities. This table detaches the creation of DQRules from specific instances of a DQEntity. This table allows a single DQRule to be applied many times to many DQEntities.
Table	Ruleset	This table allows you to create a new Ruleset which can be used to create a set of DQ Rules that will be executed as a single DQ Rule
Table	RuleExpression	This table is used to create DQ Rules that will be evaluated against a T-SQL expression
Table	RuleHarmonization	This table is used to create DQ Rules that can be used to harmonize the data being evaluated
Table	RuleValueCorrection	This table is used to create DQ Rules that replace a 'SourceValue' with a 'PreferredValue'.
Table	RuleReference	This table is used to create DQ Rules that ensure all values being evaluated exist in either a specified external reference table or a DQReferenceList (i.e. a set of values

		added to the AppReferenceLists table)
Table	RuleProfiling	This table is used to create DQ Rules that can profile incoming data
Schema: **DQ**	**This schema holds the tables, functions and stored procedures that manage the data quality rule executions.**	
Table	DataQualityHistory	This table holds the summary and top-level results from the rule executions.
Table	DataQualityRowHistory	This table holds the row level information captured by each of the data quality rules. This table uses a Data Quality Engine generated RowId to identify each failed row. This RowId is created by the DQE as a standard keying mechanism to record row information. This table does not make much meaningful sense without the associated records in the [DQ].[DataQualityPrimaryKeyValues] table.
Table	DataQualityPrimaryKeyValues	This table holds the actual primary keys of all the records logged in [DQ].[DataQualityRowHistory].
Table	RuleExecutionHistory	This table holds a record of each rule (i.e. SQL statement) constructed and applied to the data.

Stored Procedure	sApplyDQRuleExpression	Takes an expression rule code and applies it based on the configuration.
Stored Procedure	sApplyDQRuleHarmonization	Takes a harmonization rule code and applies it based on the configuration
Stored Procedure	sApplyDQRuleProfiling	Takes a profiling rule code and applies it based on the configuration
Stored Procedure	sApplyDQRuleReferences	Takes a reference rule code and applies it based on the configuration
Stored Procedure	sApplyDQRuleValueCorrect	Takes a value correction rule code and applies it based on the configuration
Stored Procedure	sExecuteJobData QualityEngine	Executes a SQL Agent Job based on an input parameter. This procedure contains a call-out to [DQ].[StartAgentJobAndWait].
Stored Procedure	StartAgentJobAndWait	Utility procedure to manage the execution of SQL Agent jobs.
Stored Procedure	sExecuteSSISData QualityEngine	Executes a SSIS catalog stored project based on an input parameter
Stored Procedure	sExecuteStandAloneRule	Executes an individual rule based on a rule code input.
Stored Procedure	sGetDomainEntities	Dual purpose creates a copy of a table if specified in the configuration & Outputs a list of active entities within a specified DQDomain. This routine contains a call out to [DQ].[sLoadCleanseEntity].
Stored Procedure	sLoadCleanseEntity	Utility procedure to manage the creation of a cleansing entity.

Stored Procedure	sGetExecutionSequences	Gets a list of execution sequence numbers within the DQDomain.
Stored Procedure	sGetDomainEntityExecution SequenceCount	Gets a count of rules within an execution sequence number.
Stored Procedure	sGetEntityDQTasks	Get list of rules for the entity
Stored Procedure	sInsertDataQualityHistory	Utility procedure to manage the insert of data into DQ.DataQualityHistory
Stored Procedure	sInsertDataQuality RowHistory	Utility procedure to manage the insert of data into DQ.DataQualityRowHistory
Stored Procedure	sInsertPrimaryKeyValues	Utility procedure to manage the insert of data into DQ.DataQualityPrimaryKeyValues
Stored Procedure	sInsertRuleExecutionHistory	Utility procedure to manage the insert of data into DQ.RuleExecutionHistory
Function	fn_ParseText2Table	Utility function.Public function used by DQE for the purposes of splitting a string and presenting the resulting value as an indexed table.
Function	fnRemoveMCharacters	Sample function included to illustrate the functionality of the SpecialOperation Harmonization rule.
Function	fnRemoveSpecialCharacters	Sample function included to illustrate the functionality of the SpecialOperation Harmonization rule.

6 Executing DQE Rules and Domains

DQE will contain a number of routines that will allow it to be execute either a single rule or multiple rules that can be applied in a specified sequence.

7 Reviewing the DQE execution activities

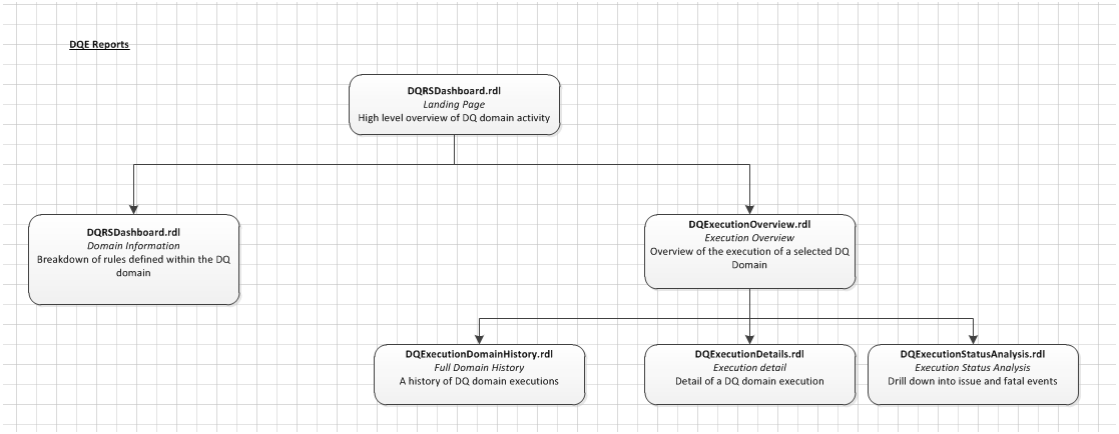
The DQE will log the following types of activity

- Data Processing logs: Each step within the DQE code is recorded together with a distinct LoadId, Name, Stored Procedure or SSIS package name, Start time, end time and the success status
- DQE Test Result logs: Each DQ Rule generates a test output together with descriptions, test results and the number of records identified
- DQE SQL logs: Most activities within the DQE are based on dynamic SQL. Each ‘chunk’ of actual SQL produced is logged.

All of the auditing information will be linked through a common identified.

8 BI Layer

A generic set of sample SSRS reports have been developed to report on DQE activities. The reports provide a navigable path through data produced by the DQE. These reports are included for illustration purposes. The hierarchy of reports is shown below



8.1 DQRS Dashboard (DQRSDashboard.rdl)

A landing page dashboard has been created to provide a number of high-level graphs. The dashboard provides summary information about

- Domain Information: The number of active DQ Rules within each DQ Domain.
- Execution Fatals: The count of errors encountered in the last execution of each DQ Domain
- Execution Information: The count of full domain executions for each DQ Domain
- Execution Information: The count of stand-alone executions for each DQ Domain

Each graph can be used to drill-down to a lower level of detail.

8.2 DQ Domain Overview (DQDomainOverview.rdl)

This report provides a breakdown of all of the rules configured within the selected DQ Domain.

8.3 DQ Execution Overview (DQExecutionOverview.rdl)

This report can be used to spot issues and errors in the previous executions. This report can be used to quickly identify where errors and issues are being detected and can be used to drill-into more detail.

The report also contains a summary table of recently encountered issues and errors.

8.4 DQ Execution Overview (DQExecutionOverview.rdl)

This report can be used to spot issues and errors in the previous executions. This report can be used to quickly identify where errors and issues are being detected and can be used to drill-into more detail.

The report also contains a summary table of recently encountered issues and errors.

8.5 DQ Execution Details (DQExecutionDetails.rdl)

This report can be used to view the detail of a selected execution. This report provides a list of all DQ rules executed within the domain together with any error messages.

8.6 DQ Status Analysis (DQExecutionStatusAnalysis.rdl)

This report can be used to view the details of any issues or fatal events detected during the execution of the selected DQ Domain.

8.7 DQ Execution Domain History (DQExecutionDomainHistory.rdl)

This report provides a list of all DQ domain executions at a DQ Rule level.