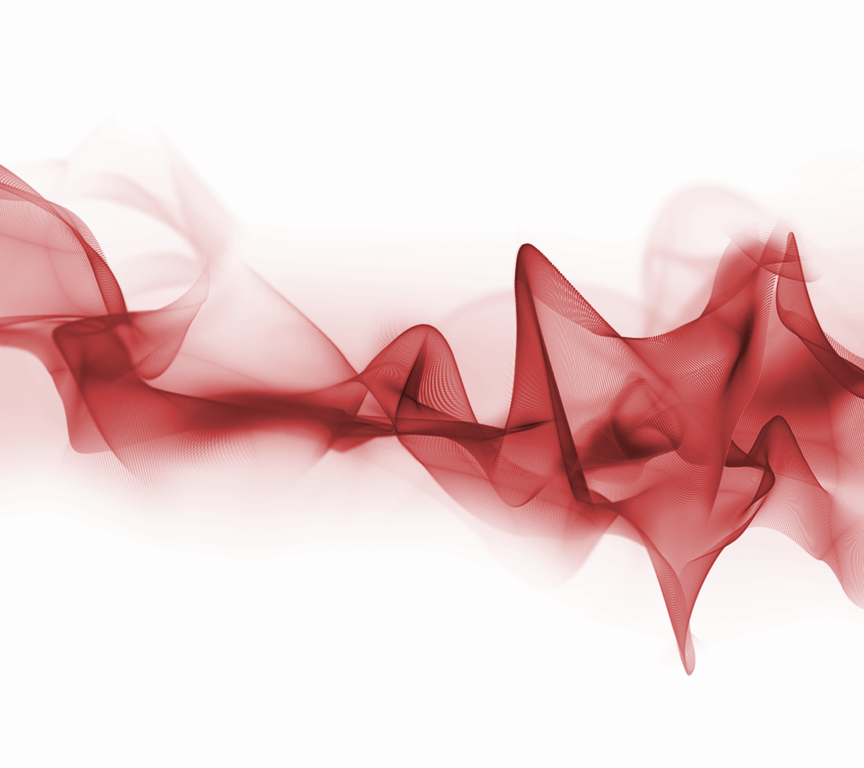
Data Quality & Data Warehouse Programme

Data Warehouse Design Principles

V1.0



© Perspicacity Ltd 2019

|  |  |
| --- | --- |
| Document Control | |
| Version | V1.0 |
| Created | 2019.01.12 |
| Updated | 2019.04.01 |
| Original Author | Matthew Bishop, Perspicacity Ltd |
| Contact | [matthew.bishop@perspicacityltd.co.uk](mailto:matthew.bishop@perspicacityltd.co.uk) / 07545 878906 |

© Perspicacity Ltd 2019

This work product is licensed under the creative commons attribution 4.0 license. This allows worldwide, royalty-free, non-exclusive rights to share, distribute and create derivative work products (including for commercial gain) as long as the terms of the license are adhered to. Full details of the terms can be found at:

<https://creativecommons.org/licenses/by/4.0/legalcode>

It is part of an open source library which can be found at <https://github.com/perspicacity-ltd/NhsDataWarehouse>

Perspicacity's other open source offerings can be found at <https://github.com/perspicacity-ltd>

**Table of Contents**

[1 NHS Data Warehouse Standard - Introduction 4](#_Toc6313124)

[2 Server instances 5](#_Toc6313125)

[3 Optimisation & Integrity 6](#_Toc6313126)

[4 Documentation 7](#_Toc6313127)

[5 XXX\_Datawarehouse\_Staging database 8](#_Toc6313128)

[6 XXX\_Datawarehouse\_Lookup database 10](#_Toc6313129)

[7 XXX\_Datawarehouse database 12](#_Toc6313130)

[8 Reporting database 14](#_Toc6313131)

[9 Reporting\_History database 16](#_Toc6313132)

[10 Resilience 17](#_Toc6313133)

[11 Security model 18](#_Toc6313134)

[12 Development – test – release cycle 19](#_Toc6313135)

# NHS Data Warehouse Standard - Introduction

This document presents a standard for building bespoke NHS data warehouses. It is a part of a repository that can be found at <https://github.com/perspicacity-ltd/NhsDataWarehouse> and initially will contain this standards document, but may expand to a standard codebase in the future.

## A little bit about the NHS Data Warehouse standards

* This set of standards emerged through the development of bespoke Data Warehouse products in multiple acute NHS trusts
* It came about using SQL server, although most of the principles are applicable to any database technology
* Adding a measure simply requires a table to hold the Data Quality detail data, a view for the SSRS report, and a Stored Procedure to update the data

## A little bit about Perspicacity Ltd

Perspicacity provides decision support consultancy, coaching, & development to the NHS. They are passionate about reducing the cost of software development to the NHS and aspire to create an active community of NHS and commercial organisations. They all share a common goal of improving the quality and efficiency of patient care through better, and more informed, decision making.

Open source helps the healthcare community to do this by sharing software development, learning from each other, and help software meet the needs of every organisation without being constrained to a single solution or paying for the same piece of work over and again across different organisations.

Although these Data Warehouse open source products are suitable for any organisation, healthcare or not, they are here as a result of wanting to freely share Perspicacity's collective products and ideas across the NHS and to widen the benefit of good, but usually locally isolated, projects further.

Perspicacity's open source offerings can be found at <https://github.com/perspicacity-ltd>

If you'd like to find out more, please contact Matthew Bishop on 07545 878906 or matthew.bishop@perspicacityltd.co.uk

# Server instances

## Production environment

### *[your\_SQL\_instance]* will present as a single production instance *[on a SQL cluster with 1 active node and 1 passive node]*

### If the production instance becomes CPU or memory bound, the production cluster can later be reconfigured to have both nodes active. A second instance, pid-sql2 will need to be set up to facilitate this, with the following characteristics:

#### The performance load on each instance will be balanced by manually choosing which instance objects sit within

#### A series of logical views, functions and stored procedures will exist between Pid-sql and Pid-sql2 pointing to the materialised objects on the other instance

#### Any configuration tables, functions and stored procedures that administrate the load balancing will exist on Pid-sql

## Test environment

### Pid-sql-test will present as a single production instance on a SQL cluster with 1 online node and 1 offline node

### Pid-sql-test will be hosted on the same cluster as Pid-sql

### If the production cluster is reconfigured to have both nodes active, the test environment will also be reconfigured to have 2 active nodes. A second instance, pid-sql-test2 will need to be set up to facilitate this, with the following characteristics:

#### The performance load on each instance will be balanced by manually choosing which instance objects sit within

#### A series of logical views, functions and stored procedures will exist between Pid-sql-test and Pid-sql-test2 pointing to the materialised objects on the other instance

#### Any configuration tables, functions and stored procedures that administrate the load balancing will exist on Pid-sql

## Development environments

### Each developer will have a development instance in the machine name range Pid-sql-devXX

### Pid-sql-devXX will be hosted on the same SQL cluster as the production e

## Transactions environments

### Any transactional database will not be created in the Pid-sql environment – instead they will be created in the clinical SQL systems environment in conjunction with the IT infrastructure team

# Optimisation & Integrity

## Entity Integrity

### All tables must have a primary key

### Where this is not possible, e.g. some staging tables, there should be an identity field based primary key to uniquely identify each record and optimise record retrieval

## Referential Integrity

### Foreign keys will only be enforced in the transactions environments, as the source systems outside of our control may not enforce referential integrity.

### Violations to referential integrity will be managed by DQ reporting

### Any instance of a referential integrity issue should be added to the DQ script and notified to the DQ team

## Constraints

### Foreign key constraints are as per referential integrity

### Default values will be used wherever a default value exists

### Value range constraints will only be enforced in the transactions environments, as the source systems outside of our control may not enforce the value ranges. Violations to value constraints will be managed by DQ reporting

### Any instance of a value constraints issue should be added to the DQ script and notified to the DQ team

## Indexing

### Indexes, other than the primary key, will be built as required. You must be wary of potentially negative implications of building indexes and test any indexes you build for performance issues

### It is recommended that foreign key fields, or any fields used in a join, are indexed

### It is recommended that fields used in filters are tested for performance improvements from indexing

## Develop to avoid Primary Key violations

### The development style of ensuring integrity of a primary key is mandatory

### Queries using multiple tables should start off (initialise) with an insert (or select into) statement using only fields from the “driving” table

### The primary key from the “driving” table should be included in the initial statement

### Data from subsequent tables should then be updated against the initial table – this will ensure that the primary key of the initial table is never violated

### Subsequent updates should be tested for duplications and omissions against the foreign key. This is most simply done by having a comment block with SELECT COUNT(\*) FROM [each table] so that variations in the rowcount can easily be identified as each table used in the FROM clause is added in

### Evidence of testing for primary key violations will be required for the release of new assets into the test / production environments

# Documentation

## Where will the conceptual specification be documented and by whom

### Explicit (external) documentation

## Where will the functional specification be documented and by whom

### Explicit (external) documentation

## Where will the technical specification be documented and by whom

### Implicit (written in comments in procedures) documentation should be used in every script / object in all databases

### A plan for your script should be decided and the commenting written before writing the actual SQL code – this will ensure that all scripts have the basic minimum documentation

### Monitored via weekly & pre-release code reviews

# XXX\_Datawarehouse\_Staging database

## Content

### This database contains exact copies of source data, plus any calculated fields that need to be done by the source system

### Each table must have a primary key, as per the source primary key

### Data types are as close as possible to the source data types, but will aim to exceed to source specification if a direct match cannot be found

### The only fields allowed in the staging database are those that exist in the corresponding source table/view or provenance fields

### Fields that require a calculation to be done in the source system (e.g. those that require the use of a database function that exists in the source schema) are acceptable

### The staging database tables must contain all the relevant provenance fields

## Programmability

### Extract Interfaces

#### Extracts will use SSIS for the extraction where possible

#### SSIS data destinations should use fast load where possible

#### Where using SSIS is not possible, extracts will use stored procedures

#### Stored procedures for extracts will be kept in the XXX\_Datawarehouse\_Staging database (i.e. in the database “pulling” the data)

### Bulk and Incremental data loads will be facilitated

#### Bulk, Incremental and Final datasets will be contained within their own schema (e.g. PAS\_LIVE\_*Bulk*, PAS\_LIVE\_*Incremental*, PAS\_LIVE\_*Final*)

#### A parent schema will exist to represent the collection of bulk and incremental extractions (e.g. PAS\_LIVE)

#### The parent schema will contain views to the *Final* schema tables

#### *Bulk* extracts will be stored in the *Bulk* datasets and replicated into the *Final* datasets upon initial *bulk* extraction

#### *Incremental* extracts will facilitate a push (i.e. based on detected changes in the source system) and a pull (i.e. based on a “session” table of records requested to be updated) mechanism

#### *Incremental* extracts will be stored in the *Incremental* dataset and then applied against the *Final* dataset

#### All *incremental* datasets extracted since the last *bulk* extract will be kept

### Environment variants of the source system will co-exist in the staging environment

#### Each variant environment (live, reporting, QA, test etc) will be contained within their own schema (e.g. PAS\_*LIVE*, PAS\_*QA*, PAS\_*REPV*, PAS\_*REPV*\_Bulk, PAS\_*REPV*\_Incremental, PAS\_*REPV*\_Final)

#### A “switching” stored procedure will be built to configure the database to switch between different variants of the same source system using dynamic DDL

#### A configuration table will be built to drive the configuration for switching between different variants of the same source system

#### DML triggers will be set up on the configuration table to ensure the “switching” stored procedure is run to bring the database DDL into line with the configuration table.

### Each Stored Procedure & SSIS package will have effective error handling routines

#### All stored procedures will use the Try… Catch construct to capture error messages

#### All target output tables that are being repopulated will perform their work on a “working” table and use the sp\_rename method to swap the “working” table for the target output table. This will minimise the downtime of the target output table and allow reporting from the target output table to continue whilst it’s “working” table is being updated

#### “Working” tables will have the same name as the target output table, but appended with “\_work” (e.g. PAS\_REPV\_Bulk.APPOINTMENTS\_work)

#### All packages and SP’s will create an audit record, in a SQL table, of either successful or failed execution for use by subsequent procedures

## Provenance fields

### *UpdatedBy* – a delimited string indicating the procedure that has updated the staging record

### *LastUpdated* - the start datetime (in ISO8601 format) of the last procedure that has updated the record

## Nomenclature

### Table names are exactly as the source table name (and in upper case if from ORACLE)

# XXX\_Datawarehouse\_Lookup database

## Content

### This database contains reference data and metadata that is applicable across all databases and source systems represented in the data warehouse environments

### There will be a schema for each source system (including external source systems e.g. TRUD, HSCIC etc)

### There will be a *Lookup* schema containing multi-system lookup tables that will allow for the resolution of metadata when a source table contains concatenated information from more than 1 source system

### Each multi-system reference data table must contain the following fields:

#### *LookupID* – a unique identity field, also to be programmed as the primary key, to uniquely identify each multi-system lookup record

#### *SourceSystem* – the name of the source system that the reference data item comes from. If the reference data comes from a function then write the name of the function

#### *SourceCode* – the unique id from the source system

#### *SourceDescription* – the description of the *SourceCode* value from the source system

#### *DestinationType* – the name of the destination system that the reference data will be output to

#### *DestinationCode* – the unique id to output to the destination system

#### *DestinationDescription* – the description to output to the destination system

## Programmability

### Transform & Load Interfaces

#### Transform and load will, preferably, use stored procedures and functions. SSIS packages can be used where a specific piece of functionality is not easily available in TSQL

#### The data source for Transform and Load packages will be XXX\_DataWarehouse\_Staging, manually entered reference data / metadata or externally sourced reference data / metadata

#### Stored procedures and functions for transform and load will be kept in the XXX\_Datawarehouse\_Lookup database (i.e. in the database “pulling” the data)

### Functions

#### Functions to categorise data will only output metadata. A corresponding reference table will be created in XXX\_Datawarehouse\_Lookups

### Each Stored Procedure & SSIS package will have effective error handling routines

#### All stored procedures will use the Try… Catch construct to capture error messages

#### All target output tables that are being repopulated will perform their work on a “working” table and use the sp\_rename method to swap the “working” table for the target output table. This will minimise the downtime of the target output table and allow reporting from the target output table to continue whilst it’s “working” table is being updated

#### “Working” tables will have the same name as the target output table, but appended with “\_work” (e.g. PAS\_REPV\_Bulk.APPOINTMENTS\_work)

#### All packages and SP’s will create an audit record, in a SQL table, of either successful or failed execution for use by subsequent procedures

## Provenance fields

### *LastUpdated* - the start datetime (in ISO8601 format) of the last procedure that has updated the record

## Nomenclature

### Table & Field names will be CamelCase, underscores will be used after an UPPERCASE acronym. There will be no pluralisation used in table names

### The following objects will be prefixed with their object type:

#### vw – views

#### usp – stored procedures

#### fn – scalar functions

#### tvf – table valued functions

#### pk\_*Tblname*\_x – Primary keys for table *Tblname*

#### ix\_*Tblname*\_x – Indexes for table *Tblname*

#### df – default value constraints

# XXX\_Datawarehouse database

## Content

### This database contains source datasets, joined where there is a referential integrity in a 1:1 relationship, with additional calculated / lookup fields and renamed field names

### Calculated fields from within the dataset, including between rows / correlated subquery calculations, are permitted

### Metadata (lookups) will be resolved

### Metadata will be included alongside resolved lookup data

### National codes and mapped codes from other systems will also be resolved

### Any other 1-1 datasets within the source schema will be de-normalised (amalgamated). Violating 2nd/3rd normal form is acceptable, but not violating 1st normal form

### Referential integrity (actual or derived / fuzzy matched) to other DW tables will be included.

### Derived referential integrity tables will be contained within a *FuzzyMatching* schema

### An assurance score for derived referential integrity will also be included. The assurance score will be initially numbered using a base 10 numbering system to allow for subsequent intermediate assurance matching processes to “fit in” to the scoring system

### Data from different sources will be concatenated (unioned) into a single dataset (e.g. DVH data into PAS data, or RIS activity with outpatient activity)

### Logically deleted data, dummy data or other data of no analytical value will be filtered out of XXX\_Datawarehouse datasets

### The warehouse database tables must contain all provenance fields

## Programmability

### Transform & Load Interfaces

#### Transform and load will, preferably, use stored procedures and functions. SSIS packages can be used where a specific piece of functionality is not easily available in TSQL

#### The data source for Transform and Load packages will only be XXX\_DataWarehouse\_Staging or XXX\_DataWarehouse\_Lookup

#### Stored procedures and functions for transform and load will be kept in the XXX\_Datawarehouse database (i.e. in the database “pulling” the data)

### Functions

#### Functions to categorise data will only output metadata. A corresponding reference table will be created in XXX\_Datawarehouse\_Lookups

### Each Stored Procedure & SSIS package will have effective error handling routines

#### All stored procedures will use the Try… Catch construct to capture error messages

#### All target output tables that are being repopulated will perform their work on a “working” table and use the sp\_rename method to swap the “working” table for the target output table. This will minimise the downtime of the target output table and allow reporting from the target output table to continue whilst it’s “working” table is being updated

#### “Working” tables will have the same name as the target output table, but appended with “\_work” (e.g. PAS\_REPV\_Bulk.APPOINTMENTS\_work)

#### All packages and SP’s will create an audit record, in a SQL table, of either successful or failed execution for use by subsequent procedures

## Provenance fields

### *LastUpdated* - the start datetime (in ISO8601 format) of the last procedure that has updated the record

## Nomenclature

### Table & Field names will be CamelCase, underscores will be used after an UPPERCASE acronym. There will be no pluralisation used in table names

### The following objects will be prefixed with their object type:

#### vw – views

#### usp – stored procedures

#### fn – scalar functions

#### tvf – table valued functions

#### pk\_*Tblname*\_x – Primary keys for table *Tblname*

#### ix\_*Tblname*\_x – Indexes for table *Tblname*

#### df – default value constraints

# Reporting database

## Content

### This database contains warehouse datasets, joined between parent/child relationships and/or across different source systems, with additional calculated fields and report ready field names

### Data will be fully de-normalised. Repeating groups and non-atomic / aggregated values are acceptable, 1st normal form can be violated

### Data between source systems (e.g. PAS activity data with finance spend data) can or will be concatenated

### Calculated fields from outside the dataset, including aggregate calculations, can be included

### Datasets that are compliant with the national data dictionary (CDS) will be contained here

### Referential integrity (actual or derived / fuzzy matched) from the associated DW tables will be included. The assurance score for derived referential integrity will also be included

### Patient identifiable data (PID) that is relevant to reporting will be stored in separate “sibling” tables, using the primary key of the table to which they relate

### PID “sibling” tables will be contained in the PID schema and will have the same name as their sibling counterpart

### The reporting database tables must contain all provenance fields

## Programmability

### Transform & Load Interfaces

#### Transform and load will, preferably, use stored procedures and functions. SSIS packages can be used where a specific piece of functionality is not easily available in TSQL

#### The data source for Transform and Load packages will only be XXX\_DataWarehouse

#### A *DW* schema will be created for all transform and load interface objects

#### Stored procedures and functions for transform and load will be kept in the Reporting database (i.e. in the database “pulling” the data) under the *DW* schema

### User / Reporting Interface

#### Reporting will be via “stubs” created as stored procedures or functions

#### An *UI* schema will be created for all user / reporting interface objects (“stubs”)

#### Each SP or function in the UI schema will have an @PID parameter, which will pass the username of the requesting user

#### The presentation of Patient Identifiable Data (PID) will be controlled using groups in the active directory. The username passed to the @PID parameter of each SP or function will be used in an LDAP openquery to determine whether the user is permitted to see anonymised, pseudonymised or clear patient data

#### Stored procedures and functions for the user / reporting interface will be kept in the Reporting database (i.e. in the database “pulling” the data) under the *UI* schema

#### Where variants of the same dataset are required, they will provided by the same stored procedure / function to minimise the number of objects in the database (and, therefore, minimise the risk of duplicating objects if the SSMS object explorer has a minimal number of objects to look through)

#### Where there are variants of the same dataset provided by the same stored procedure, the required output will be differentiated using parameters (e.g. @ReportVariant)

### Functions

#### Functions to categorise data will only output metadata. A corresponding reference table will be created in XXX\_Datawarehouse\_Lookups

### Each Stored Procedure & SSIS package will have effective error handling routines

#### All stored procedures will use the Try… Catch construct to capture error messages

#### All target output tables that are being repopulated will perform their work on a “working” table and use the sp\_rename method to swap the “working” table for the target output table. This will minimise the downtime of the target output table and allow reporting from the target output table to continue whilst it’s “working” table is being updated

#### “Working” tables will have the same name as the target output table, but appended with “\_work” (e.g. PAS\_REPV\_Bulk.APPOINTMENTS\_work)

#### All packages and SP’s will create an audit record, in a SQL table, of either successful or failed execution for use by subsequent procedures

## Provenance fields

### *LastUpdated* - the start datetime (in ISO8601 format) of the last procedure that has updated the record

## Nomenclature

### Table & Field names will be report ready (proper case, with spaces)

### The following objects will be prefixed with their object type:

#### vw – views

#### usp – stored procedures

#### fn – scalar functions

#### tvf – table valued functions

#### pk\_*Tblname*\_x – Primary keys for table *Tblname*

#### ix\_*Tblname*\_x – Indexes for table *Tblname*

#### df – default value constraints

# Reporting\_History database

## Content

### This database contains change data between source extracts, including calculated fields that need to be done by the source system. This will provide a mechanism to record updates and deletions to records that are not recorded in the source system audit trails

### This database contains change data between the Reporting database transform & load runs. This will provide a mechanism to record updates and deletions to records that are not recorded elsewhere

### This database will not be used for any automated reporting, but only to fault trace errors or provide audit data

## Programmability

### Transform & Load Interfaces

#### Transform and load will, preferably, use stored procedures and functions. SSIS packages can be used where a specific piece of functionality is not easily available in TSQL

#### The data source for Transform and Load packages will only be the *Reporting* database

#### Stored procedures and functions for transform and load will be kept in the Reporting\_History database (i.e. in the database “pulling” the data)

### Each Stored Procedure & SSIS package will have effective error handling routines

#### All stored procedures will use the Try… Catch construct to capture error messages

#### All target output tables that are being repopulated will perform their work on a “working” table and use the sp\_rename method to swap the “working” table for the target output table. This will minimise the downtime of the target output table and allow reporting from the target output table to continue whilst it’s “working” table is being updated

#### “Working” tables will have the same name as the target output table, but appended with “\_work” (e.g. PAS\_REPV\_Bulk.APPOINTMENTS\_work)

#### All packages and SP’s will create an audit record, in a SQL table, of either successful or failed execution for use by subsequent procedures

## Provenance fields

### No provenance will be directly recorded – provenance data will be a direct copy of provenance in the source data table

## Nomenclature

### Table names are exactly as the source table name

# Resilience

## Alerting

### All jobs should have an alerting mechanism to indicate full or partial failures of the components contained within. Notifications should be sent to the information team, coding team, DQ team and development team

## Data Backup

### This should be as per the IT strategy

## Schema Backup

### Weekly scripts should be generated for databases, agent jobs, security and linked servers and stored in the GIT repository

## Failover testing

### This should be as per the IT strategy

## Cover & OOH Cover

### This should be documented in the BI policy

## DR Testing

### This should be as per the IT strategy

# Security model

## Authentication between databases

### Where a database system is hosted on site, or on a federated domain, dedicated SQL service accounts should be used where possible to connect to an external database. If this is not possible use service accounts and where this is not possible use user accounts

## How will users be granted access

### Users will be granted access either directly on the tables that hold data, or using LDAP queries to the AD where specific permissions are required (see section 7.2)

## How will anonymization / pseudonymisation be administered

### As per section 7.2

## Who is the custodian

### The appointed DBA within the information team establishment is the custodian

## Protection from hack attack vectors

### For stored procedures and functions with writeback functionality, a function will be created to check all string parameters using dynamic SQL or a where clause to minimise the risk of SQL injection attacks

# Development – test – release cycle

## Development environment

### Each user will have their own development instance on SQL server with established mechanism to propagate schematic and data changes to development environments.

## DBA monitoring of unauthorised changes

### The DBA will use the Redgate SQL monitor tool to detect unauthorised changes to databases

## Version control

### All developers will use a trust GIT, SVN or MSVSS repository and Redgate source control software

### Lookup tables should be added to source control

## Change scripts and Rollback process

### All developers will use the Redgate SQL compare and Data compare tools to generate change scripts to be applied to test and production databases

### Change scripts will be saved in the trust GIT repository

## Release process and authorisation

### All releases to live to be done only by the DBA or an authorised deputy

### Peer QA will be required to for a change script to proceed to the test environment

### DBA QA will be required for a change script to proceed to the production environment