Oracle ® Enterprise Data Quality

Architecture Guide

Version 9.0

January 2012

ORACLE

Table of Contents

1 Introduction

This document provides a high-level introduction to the architecture and implementation of OEDQ. It describes the software components of the system and the workflow of its major operations, as well as the relative resource demands made by each element.

To avoid complicating this high-level picture, this document does not discuss in-depth technical details or performance optimization techniques.

1.1 Who is this document for?

This document is designed for readers with a technical background who need to know how OEDQ works and what demands it will place on a platform once deployed. Readers are also assumed to have a working knowledge of SQL RDBMS and Java application servers.

1.2 OEDQ concepts

Oracle Enterprise Data Quality (OEDQ) is a Data Quality product that provides integrated data profiling, cleaning, parsing and matching functionality. The core elements of OEDQ are:

snapshots A *snapshot* is a captured copy of external data, stored within the OEDQ repository.

processes A *process* specifies a set of actions to be performed on some specified data. It is made up of a series of *processors*, each specifying how data is to be handled and the rules that should be applied to it. A process may produce:

- *Staged data*: data or metrics produced by processing the input data and choosing to write output data to the results database.
- *Results data*: metric information summarizing the results of the process. For example, a simple validation process may record the number of records that failed and the number of records that passed validation.

processors A *processor* is a logical element which performs some operation on the data. Processors can perform statistical analysis, audit checks, transformations, matching, or other operations. Processors are chained together to form processes.

reference data *Reference data* consists of lists and maps that can be used by a processor to perform checking, matching, transformations and so on. Reference data can be supplied as part of OEDQ or by a third party, or can be defined by the user.

staged data *Staged data* consists of data snapshots and data written by

processes, and is stored within the results schema.

For more details of these and other concepts please see the 'Concepts' section of the Online Help.

2 High level architecture

Oracle Enterprise Data Quality (OEDQ) is a Java Web Application with a client-server architecture. It consists of:

- Web Start graphical user interfaces (client applications),
- a business layer (the server), which is built around a Java servlet engine, and
- a SQL RDBMS system (the data repository).

Additionally, it may communicate with other databases to read or write data. (See Figure 1: Overview of the system).

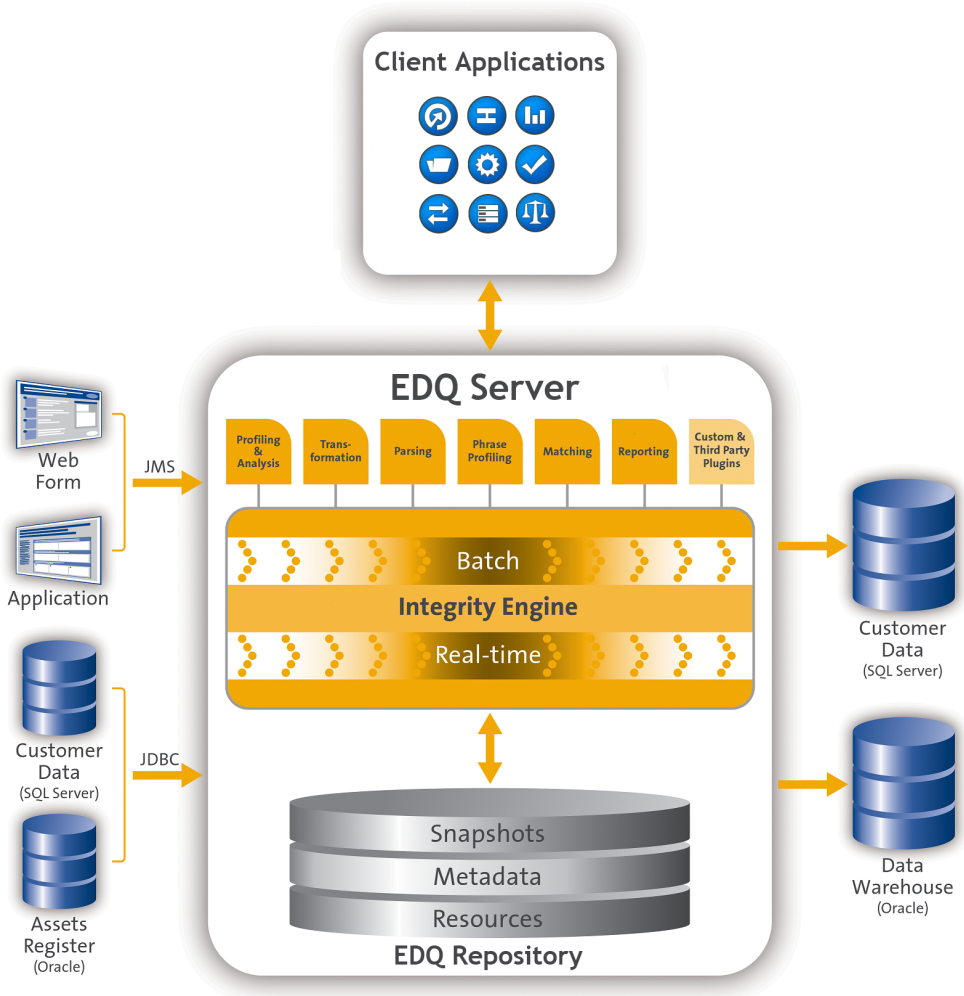

Figure 1: Overview of the system

The components of OEDQ may be deployed over one or many computers. For example, OEDQ can be installed as a standalone instance on a single computer, such that the client, server and repository are all present on the same machine. Alternatively, OEDQ can be deployed in a multi-user capacity, with the web application server and repository installed on a single, networked machine and the client on several user workstations. It is also possible for the repository, and even the individual databases which form the repository, to reside on a different machine from the application server, if required.

3 Software components

As has already been seen, OEDQ is a client-server application which consists of several software components. The following sections detail the operation of these components and their data storage, data access and I/O requirements.

3.1 Graphical user interface

The OEDQ Director graphical user interface (GUI) is a Java Web Start application. It is a rich Java GUI that allows users to:

- configure data capture and processing,
- initiate, monitor and cancel data processing,
- browse results, and
- receive notification of events occurring on the server, such as progress updates and configuration changes made by other users.

The GUI is installed, and runs on, the client computer or computers within the system.

3.1.1 Data storage

The client computer only stores user preferences for the presentation of the GUI; all other information is stored on the OEDQ server.

3.1.2 Network communications

The GUI and the business layer communicate over either an HTTP or HTTPS connection, as determined by the GUI configuration on start-up. For simplicity, this connection will be referred to as 'the HTTP connection' in the remainder of this document.

3.2 SQL RDBMS data storage

OEDQ uses a repository which contains two database schemas: the director schema and the results schema.

3.2.1 Director schema

The director schema stores configuration data for OEDQ. It is generally used in the typical transactional manner common to many web applications.

Data storage

Only a small amount of data is held in this schema. For a simple implementation, this is likely to be in the order of several megabytes. In the case of an exceptionally large OEDQ system, the storage requirements of the Director schema might reach 10 Gb.

Data access

Access to the data held in the director schema is typical of configuration data in other RDBMS applications. Most database access is in the form of read requests, with relatively few data update and insert requests.

3.2.2 Results schema

The results schema stores snapshot, staged and results data. It is highly dynamic, with tables being created and dropped as required to store the data handled by processors running on the server. Temporary working tables are also created and dropped during process execution to store any working data that cannot be held in the available memory.

Data storage

The amount of data held in the results schema will be very variable over time, and data capture and processing can involve gigabytes of data. Data may also be stored in the results database on a temporary basis, whilst a process runs.

Data access

The results schema shows a very different data access profile to the director schema, and is extremely atypical of a conventional web-based database application. Typically, tables in the results schema are:

- created on demand;
- populated with data using bulk JDBC APIs;
- queried using full table scans to support process execution;
- indexed;
- queried using complex SQL statements in response to user interactions with the GUI;
- dropped when the process or snapshot they are associated with is run again.

The very dynamic nature of this schema means that it must be handled carefully. For example, because large volumes of data being read from and written to its tables, facilities such as the Oracle recycle bin have a negative impact on both storage requirements and performance.

3.3 The business layer

The business layer fulfils three main functions:

- It provides the API that the GUI uses to interact with the rest of the system.
- It notifies the GUI of server events that may require GUI updates.
- It runs the processes which capture and process data.

3.3.1 Data storage

The business layer stores configuration data in the director schema, and working data and results in the results schema.

3.3.2 Network communications and CPU load

When passing data to and from the GUI the business layer behaves in a manner common to most traditional Java Web Applications. The business layer makes small database transactions and sends small volumes of information to the front end using HTTPS. It is somewhat unusual in that the application front end is a rich GUI rather than a browser. The data sent to the GUI therefore consists mostly of serialized Java objects rather than the more traditional HTML.

However, when running processes and creating snapshots, the business layer behaves more like a traditional batch application. In its default configuration, it spawns multiple threads and database connections in order to handle potentially very large volumes of data, and will use all available CPU cores and database I/O capacity.

It is possible to configure OEDQ to limit its use of available resources, but this has clear performance implications.

4 Major operations

The previous sections have detailed the major software components of OEDQ. The following sections will describe some of the most significant and resource-intensive operations performed by OEDQ; data capture, general data processing, match processing and real-time data processing.

4.1 Data capture

The data capture process, also known as *snapshotting*, begins with retrieving the data to be captured from an external *data source*. Data can be captured from databases, text files, XML files and so on. For a comprehensive list of possible types of data source, please refer to the Data Stores topic in the Concepts section of the Online Help. Depending on the type of data source, data capture may involve:

- running a single SQL query on the source system.
- sequentially processing a delimited or fixed format file.
- processing an XML file to produce a stream of data.

As the data is retrieved, it will be processed by a single thread. This involves:

- assigning an internal sequence number to each input record. This is usually a monotonically increasing number for each row.
- batching the rows into *work units*. Once a work unit is filled, it is passed into the results database *work queue*.

The database work queue is made up of work requests — mostly data insertion or indexing requests — to be executed on the database. The queue is processed by a pool of threads which retrieve work units from the queue, obtain a database connection to the appropriate database, and execute the work. In the case of snapshotting, the work will consist of using the JDBC batch API to load groups of records into a table.

Once all the data has been inserted for a table, the snapshot process creates one or more indexing requests and adds them to the database work queue. At least one indexing request will be created per table—to index the unique row identifier—but depending on the volume of data in the snapshot and the configuration of the snapshot process, other columns in the captured data may also be used to generate indexes into the snapshot data.

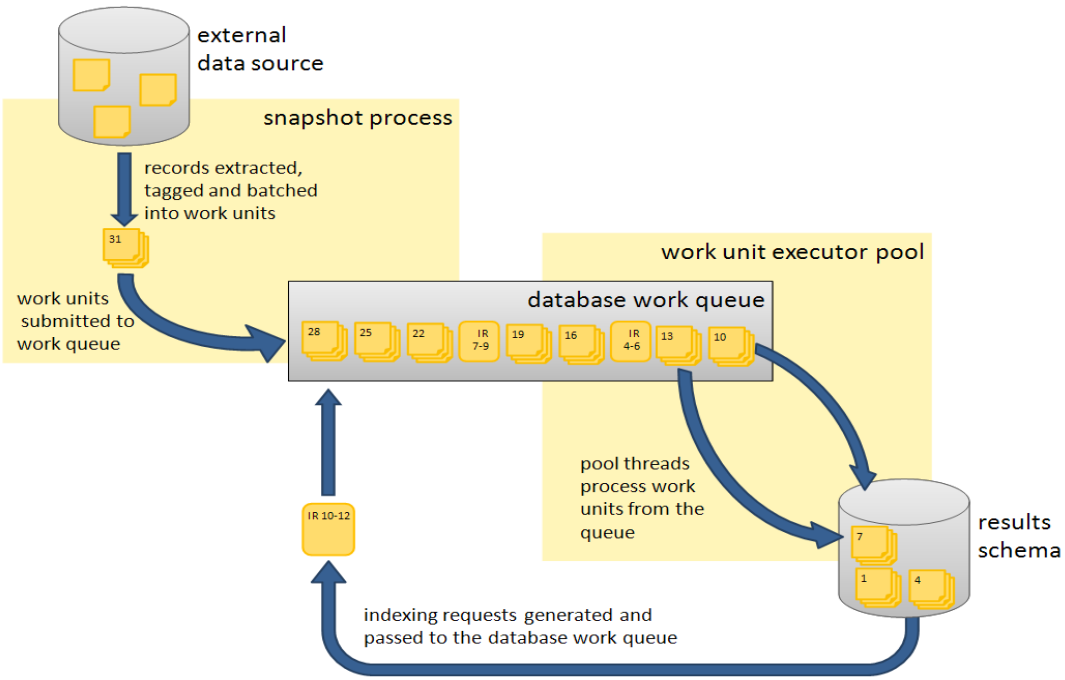

Figure 2: The data capture process

4.1.1 Network communications and CPU load

Snapshotting is expected to generate:

- I/O and CPU load on the machine hosting the data source whilst data is read;
- CPU load on the web application server caused by the snapshot process reading data and grouping it for insertion;

- I/O and CPU load on the web application server, caused by the database work unit executor threads;
- a significant amount of I/O on the machine hosting the OEDQ results database as the data is inserted into a new table;
- followed by a significant amount of I/O and some CPU load on machine hosting the results database as the data is indexed;

For example, a default OEDQ installation on a 4-core server which is taking a snapshot of 10,000 rows of data 10 columns wide would generate SQL of the following form:

1. `DROP TABLE DN_1;`
2. `CREATE TABLE DN_1 (record_id, column1, column2, …, column10);`
3. 100 bulk insert statements of the form:

```
INSERT INTO DN_1 (record_id, column1, column2, .., column10)
VALUE ( ?, ?, …, ? );
```

each taking a group of 100 parameters. The bulk inserts would be executed in parallel over four separate database connections, one per CPU core.

4. `ANALYZE TABLE DN_1 ESTIMATE STATISTICS SAMPLE 10 PERCENT`
5. And finally, eleven `CREATE INDEX...` statements, indexing each of the columns in the new table (the original ten columns, plus the record_id). The `CREATE INDEX` statements would also be executed in parallel over four database connections.

4.2 General data processing

Once the data has been captured, it is ready for processing. A process consists of a reader processor, one or more downstream processors and one or more optional writer processors. The reader processor provides the downstream processors with managed access to the data, and the downstream processors produce results data. If any writer processors are present, they will write the results of processing back to the staged data repository.

Running a process causes the web application server to start a number of *process execution threads*. The default configuration of OEDQ will start as many threads as there are cores on the OEDQ application server machine.

4.2.1 Work sharing

Each process execution thread is assigned a subset of the data to process. When the input data for a process is a data set of known size, such as snapshot or staged data, each thread will execute a query to retrieve a subset of the data, identified by the unique row IDs assigned during snapshotting. So, in our example scenario of processing 10,000 records of data on a 4-core machine, four queries will be issued against the results schema. The queries would be of the form:

```
SELECT record_id, column1, column2, … , column10
```

```
FROM DN_1
WHERE record_id > 0 AND record_id <= 2500;
```

In the case where the process is not run against a data set of known size, such as a job scheduled to run directly against a data source, records are shared amongst the process execution threads by reading all records into a queue which is then consumed by the process execution threads.

Each process execution thread is also made aware of the sequence of processors which make up the process. The process execution threads pass the records through each of the appropriate processors. As the processors work, they accumulate results that need to be stored in the results schema and, in the case of writer processors, they may also accumulate data that needs to be written to staged data. All this data is accumulated into insertion groups and added into database work units, which are processed as described in the Data capture section.

Once an execution thread has processed all its assigned records, it waits for all other process execution threads to complete. The process execution threads then enter a *collation phase* in which the summary data from the multiple copies of the process are accumulated and written to the results database, again via the results database work queue.

As a batch is processed, we expect to see:

- Read load on the results schema as the captured data is read;
- CPU load on the web application server as the data is processed;
- Significant write load on the results schema as results and staged data are written to the schema;
- Reduced CPU load as the collation phase is entered;
- A small amount of further database work as any outstanding database work units are processed and accumulated results written;
- Further write load on the results schema at the end of the collation phase, in the form of requests to index the results and staged data tables, as necessary. The size and number of the index requests will vary, depending on data volumes and system configuration.

Processes that are heavily built around cleaning and validation operations will tend to be bound by the I/O capacity of the database. Some processors consume significant CPU resource, but generally the speed of operation is determined by how quickly data can be provided from and written to the results schema.

4.2.2 All record processing

There are a number of processors, such as the duplicate profiler and the duplicate check processor, that require access to the whole record set in order to work. If these processors only had access to a subset of the data, they would be unable to detect duplicate records with any accuracy. These processes use multiple threads to absorb the input records and build

them into a temporary table. A single thread then examines all the records and performs the necessary processing. Once all the records have been examined, they are re-emitted by distributing the records amongst the various process execution threads. There is no guarantee that a record will be emitted on the same process execution thread that absorbed it.

4.3 Match processing

It is recommended that the reader is familiar with the material contained in the "Advanced Features: Matching Concept Guide" contained within the Online Help before reading this section. An understanding of the concepts involved in the OEDQ matching process will greatly aid understanding of the material presented here.

OEDQ match processors are handled in a significantly different way from the simpler processors. Due to the nature of the work carried out by match processors, multiple passes through the data are required.

A match processor is executed by treating it as a series of sub-processes. For example, consider a process which is designed to match a customer data snapshot against a list of prohibited persons. The process contains a match processor which is configured to produce a list of customer reference numbers and related prohibited person identifiers. Each data stream which is input to, or output from, the match processor, is considered to be a sub-process of the match processor. Therefore, in this example we have three sub-processes, representing the customer data input stream, the prohibited persons input stream and the output data stream of the match processor. The match processor itself forms a fourth sub-process, which effectively couples the data inputs to its outputs. Each sub-process is assigned the normal quota of process execution threads, so on a 4-core machine, each sub-process would have four process execution threads.

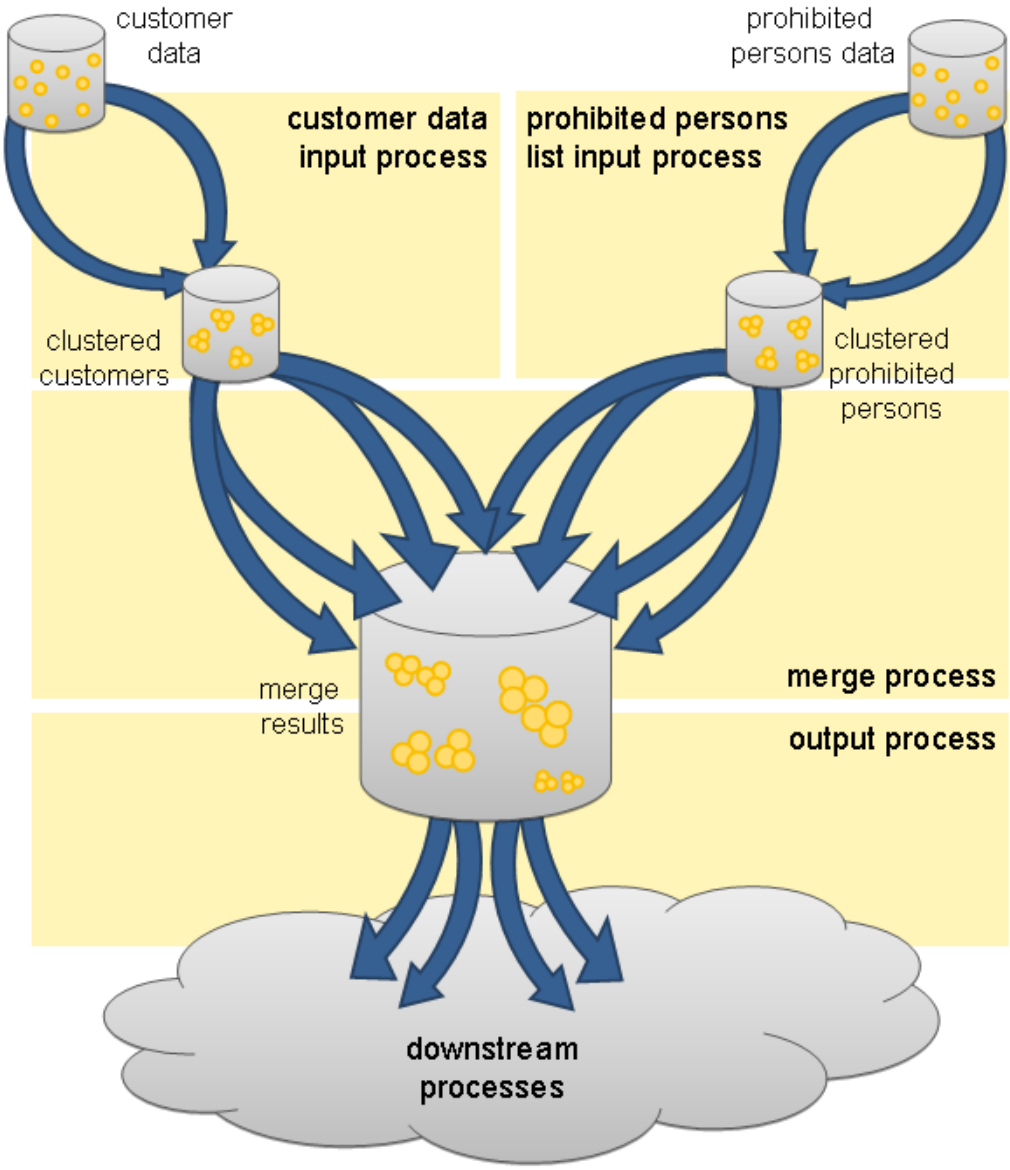

Figure 3: Match process threads

When execution of the match processor begins, the input data sub-processes run first, processing the input data. At this point, there is no work available for the match or match output sub-processes, which remain dormant. The input data sub-processes generate cluster values for the data streams and store the cluster values and incoming records in the results schema, via the normal database work units mechanism.

Once the input data sub-processes have processed all the available records, they terminate and commence collation of their sub-process results. Meanwhile, the match sub-process will become active. The match sub-process then works through a series of stages, with each process execution thread waiting for all the other process execution threads to complete each stage before they progress to the next. Each time a new stage begins, the work will be subdivided amongst the processor executor threads in a manner that is appropriate at that stage. The processing stages are:

Comparison phase	The customer data and prohibited people data is retrieved, ordered by cluster values. The data is gathered into groups of equal cluster values and then handed, via a queue, to the match process threads to compare the records. Where relationships are found between records the relationship information is written to the results schema.
Provisional grouping phase	The relationship information detected during the comparison phase is retrieved in chunks and provisional groups of related records are formed. The relationship chunks are processed in parallel by the match processor threads. These provisional groups are written back to the results database.
Final grouping phase	The provisional group table is inspected by a single thread to check for groups that have been artificially split by chunk boundaries. If any such cross-chunk groups are found they are merged into a single group.
Merged output phase	Each of the match processor threads retrieves an independent subset of the match groups and forms the merged output, merging multiple records into the single output records.

This completes the match sub-process, and so the match processor execution threads now move into their collation phase.

At this point, the sub-process associated with the output of match data becomes active. The output data is divided amongst the process execution threads for the output sub-process and passed to the processors down stream from the match processor. From this point onwards, the data is processed in the normal batch processing way.

Benchmarks and production experience have shown that the comparison phase of a match processor is one of the few OEDQ operations that is likely to become CPU bound. When anything other than very simple comparison operations are performed, the ability of the CPU to handle the comparison load limits the process. The comparison operations scale very well and are perfectly capable of utilizing all CPU cycles available to the OEDQ Web Application Server.

4.4 Real-time processing

OEDQ is capable of processing messages in real time. Currently, OEDQ supports messaging via:

- Web Services;
- JMS-enabled messaging software.

When configured for real-time message processing, the server starts multiple process execution threads to handle messages as they are received. An incoming message is handed

to a free process execution thread, or placed in a queue to await the next process execution thread to become free. Once the message has been processed, any staged data will be written to the results database, and the process execution thread will either pick up the next message from the queue, if one exists, or become available for the next incoming message.

When processing data in real time, the process will normally be run in *interval mode*. Interval mode allows the process to save results at set intervals so that they can be inspected by a user. The interval can be determined either by the number of records processed or by time limit. When an interval limit is reached, OEDQ starts a new set of process execution threads for the process. Once all the new process execution threads have completed any necessary initialization, any new incoming messages are passed to the new threads. Once the old set of process execution threads have finished processing any outstanding messages, the system directs those threads to enter the collation phase and save any results, after which the old process execution threads are terminated and the data is available for browsing.

5 Application security

Application security is incorporated in many aspects of the application architecture.

5.1 Client-server communication

The security of communication between the web application server and the client applications is determined by the configuration of the Java Application Server hosting OEDQ. The Java Application Server can be configured to use either HTTP or HTTPS. HTTPS is always used by default, with all HTTP requests automatically redirected to the HTTPS port.

5.2 Authentication

OEDQ authenticates user passwords against values held in the director database. The passwords are held in a hashed form that the application cannot reverse. This configuration is used by:

- the client user applications;
- the OEDQ web pages.

The client user applications authenticate users using a proprietary protocol over HTTPS. In addition, passwords are hashed before being sent to the server.

The web pages are secured using forms-based authentication. Again this communicates with the server over HTTPS.

Mandatory password strength enforcement can also be configured, encompassing the following criteria:

- Minimum length;
- Minimum number of non-alphabetic characters;
- Minimum number of numeric characters;

- Prevention of recent password re-use;
- Prevention of using the user name in the password.

Account security encompassing the following criteria can also be configured:

- Password expiry;
- Application behavior following failed login attempts.

5.3 Storage of secrets

OEDQ requires a number of secrets; these are stored in a number of places depending on their nature:

- Connection details for databases that OEDQ connects to in order to perform snapshots and dynamic value lookups are stored in the director schema. This includes user names, passwords, hosts names and port numbers for the database connections. Passwords are stored in an encrypted form in the director schema and decrypted by OEDQ when it needs to log into these databases. The decrypted password is not represented to OEDQ users or administrators. The encryption/decryption key for the passwords is generated randomly for each installation of OEDQ and stored in a file in the the OEDQ configuration directory. This random key generation on a per-installation basis ensures that encrypted passwords cannot be copied meaningfully between systems.
- The connection details, including the user names and passwords, for the director schema and the results schema are stored in clear text format in a file in the OEDQ configuration directory on the server.

5.4 Data segmentation

When OEDQ is being used across multiple business lines, or when several businesses are using the same system, it is important to be able to segment user access to data. OEDQ achieves this by allowing users and projects to be allocated to groups. Users can only access a project if they are members of the same group as the project. The Director user application only presents accessible projects to a given user; all other projects are invisible, and all contents, including reference data, web services and so on are unavailable to unauthorized users.

> **NOTE:** Since administrative users must be able to manage all the projects in the system, any user with permission to create projects can see all projects in the system regardless of project settings.