Oracle® Enterprise Data Quality

Architecture Guide

Release 11g R1 (11.1.1.9)

E55995-01

April 2015

This document describes the Enterprise Data Quality (EDQ) architecture.

1 Software Components

EDQ is a client-server application comprised of several Graphical User Interfaces (GUIs), a data repository, and a business layer. The following sections detail the operation of these components and their data storage, data access, and I/O requirements.

1.1 Graphical User Interfaces

EDQ provides a number of GUIs that are used to configure and operate the system. Most are Java Web Start applications, and the remainder are simple web pages. The following table lists all the GUIs:

GUI Name	Technology	Purpose
Director	Web Start	Design and test data quality processing
Server Console	Web Start	Operate and monitor jobs
Match Review	Web Start	Review match results and make manual match decisions
Dashboard	Browser	Monitor DQ KPIs and trends
Case Management	Web Start	Perform detailed investigations into data issues through configurable workflows
Case Management Administration	Web Start	Configure workflows and permissions for Case Management
Web Service Tester	Browser	Test EDQ Web Services
Configuration Analysis	Web Start	Report on configuration and perform Diffs between versions of configuration
Issue Manager	Web Start	Manage a list of DQ issues
Administration	Browser	Administer the EDQ server (users, groups, extensions, launchpad configuration)
Change Password	Browser	Change password

GUI Name	Technology	Purpose
Configuration Analysis	Web Start	Analyze project configurations, and report on differences'.

The GUIs can be accessed from the EDQ Launchpad on the EDQ server. When a client launches one of the Java Web Start applications, such as Director, the application is downloaded, installed, and run on the client machine(s). The application communicates with the server to instantiate changes and receive messages from the server, such as information about tasks that are running and changes made by other users.

As EDQ is an extensible system, it can be extended to add further user applications when installed to work for a particular use case. For example, Oracle Watchlist Screening extends EDQ to add its own user application for screening data against watchlists.

Note: Many of the GUIs are available either separately (for dedicated use) or within another GUI. For example, the Configuration Analysis, Match Review and Issue Manager GUIs are also available in Director.

1.1.1 Data Storage

The client computer only stores user preferences for the presentation of GUIs, such as Director. All other information is stored on the EDQ server.

1.1.2 Network Communications

The GUIs 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 is referred to as 'the HTTP connection' in the remainder of this document.

1.2 SQL RDBMS Data Storage

EDQ uses a repository that contains two database schemas: the Config schema and the Results schema.

Note: Each EDQ server must have its own Config and Results schemas. If multiple servers are deployed in a High Availability architecture, configuration cannot be shared by pointing both servers to the same schemas.

1.2.1 Config Schema

The Config schema stores configuration data for EDQ. It is generally used in the typical transactional manner common to many web applications: queries are run to access small numbers of records, which are then updated as required.

Data Storage

Normally, only a small amount of data is held in this schema. In simple implementations, it is likely to be in the order of several megabytes. In the case of an exceptionally large EDQ system, especially where Case Management is heavily used, the storage requirements could reach 10 GB.

Data Access

Access to the data held in the Config 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.

1.2.2 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 vary significantly 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 while a process or a job runs. In the case of a job, several versions of the data may be written to the database during processing.

Data Access

The Results schema shows a very different data access profile to the Config 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 dynamic nature of this schema means that it must be handled carefully. For example, it is often advisable to mount redo log files on a separate disk.

1.3 The Business Layer

The business layer fulfills three main functions:

- Provides the API that the GUI uses to interact with the rest of the system.
- Notifies the GUI of server events that may require GUI updates.
- Runs the processes that capture and process data.

1.3.1 Data Storage

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

1.3.2 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 the HTTP connection. This is somewhat unusual in that the application front-ends are mostly rich GUIs rather than browsers. Therefore the data sent to the GUI 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 uses all available CPU cores and database I/O capacity.

It is possible to configure EDQ to limit its use of available resources, but this has clear performance implications. For further information, see the EDQ Installation Guide and EDQ Performance Tuning Guide.

2 Product Concepts

The most important elements of EDQ are:

Element	Description
snapshot	A **snapshot** is a captured copy of external data stored within the EDQ repository.
processor	A **processor** is a logical element that 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.
process	A **process** specifies a set of actions to be performed on some specified data. It comprises 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.
job	A **job** is a configured and ordered set of tasks that may be instigated either by EDQ or externally. Examples of tasks include executions of file downloads, snapshots, processes, and exports.
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 EDQ 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, see the Concepts section of the EDQ Online Help.

3 Major Operations

This section describes some of the most significant and resource-intensive operations performed by EDQ; data capture, general data processing, match processing, and real-time data processing.

3.1 Data Capture

The data capture process 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, 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.

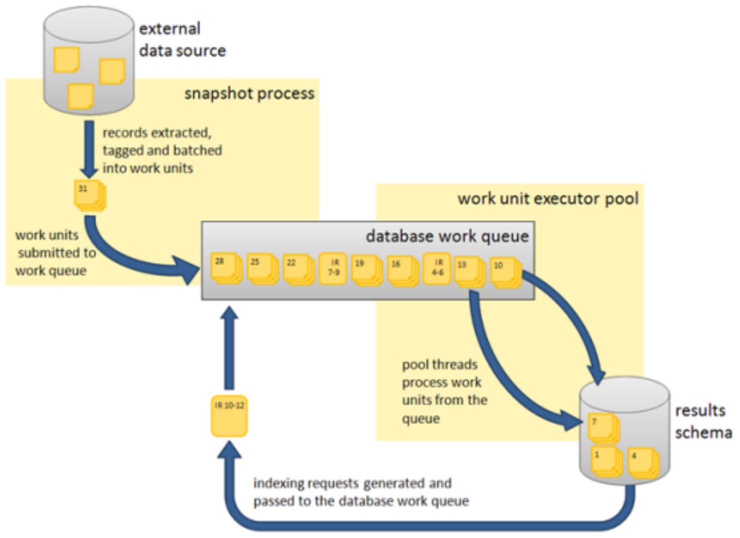
As the data is retrieved, it is processed by a single thread. This involves:

- 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 that 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 1 The Data Capture Process


external data source
snapshot process
records extracted, tagged and batched into work units
31
work units submitted to work queue
work unit executor pool
database work queue
28
25
22
IR 7-9
19
16
IR 4-6
13
10
pool threads process work units from the queue
results schema
7
1
4
IR 10-12
indexing requests generated and passed to the database work queue

3.1.1 Network Communications and CPU Load

Snapshotting is expected to generate:

- I/O and CPU load on the machine hosting the data source while 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 EDQ Results database as the data is inserted into a new table
- 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 EDQ installation on a 4-core server taking a snapshot of 10,000 rows of data 10 columns wide would generate SQL of the following form:

```
DROP TABLE DN_1;
CREATE TABLE DN_1 (record-id, column1, column2, ..., column10);
```

100 bulk insert statements of the form:

```
INSERT INTO DN_1 (record_id, column1, column2, ..., column10) VALUES ( ?, ?, ...,
? )
```

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

```
ANALYZE TABLE DN_1 ESTIMATE STATISTICS SAMPLE 10 PERCENT
```

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.

3.2 General Data Processing

Once the data has been captured, it is ready for processing. 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 EDQ will start as many threads as there are cores on the EDQ application server machine.

3.2.1 Streaming

Instead of capturing data in a snapshot and storing it in the results database (other than temporarily during collation), it can be pulled from a source and pushed to targets as a stream.

3.2.2 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 the example scenario in Section 3.1.1 describing the 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 that comprise 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 4.1 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**, during which the summary data from the multiple copies of the process are accumulated and written to the Results database by the database work queue.

The following behavior is expected during batch processing:

- 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.

3.2.3 All Record Processing

There are a number of processors, such as the Record Duplication Profiler and Duplicate Check processors, 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. 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.

3.2.4 Run Labels

During the design phase of a project, processes and jobs are typically run interactively using Director. When a job is run in Director, results will typically be written for inspection by the user so that the configuration of processes and jobs can be iterated to work optimally with the in-scope data. The amount of data to write can be controlled in Director.

However, when a job is deployed to production such detailed results are not normally required, and jobs are typically run with Run Labels, either from the Server Console or from the command line. When run with a Run Label, a job will only write the staged data and results views that the user has configured to be staged in the job, for better performance efficiency.

Note: Jobs may be designed independently of any specific source or target of data. Such jobs will normally be run with a set of command line parameters, or a stored Run Profile that sets the same parameters, that dynamically change key configuration points such as the physical source of the data to read, key processing options, and the physical source of the data to write. Such jobs need to be run with a Run Label so that the written data and results are clearly separated from other runs of the job on different data. Server Console allows users to inspect results by Run Label.

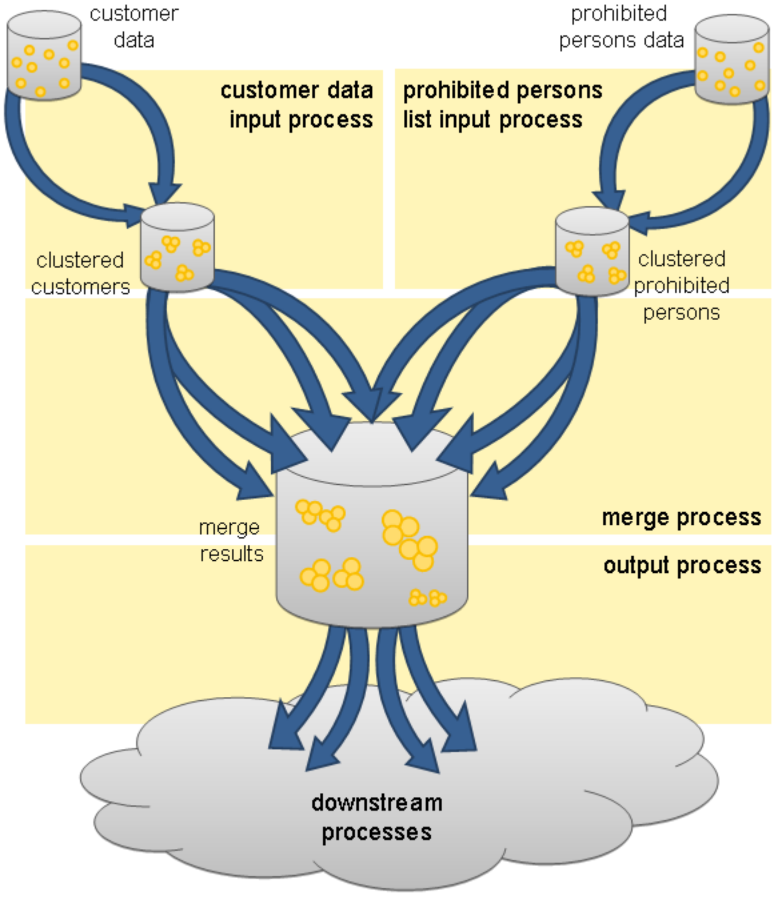
3.3 Match Processing

Oracle recommends that the reader familiarizes themselves with the material contained in the "Advanced Features: Matching Concept Guide" in the Online Help (http://www.oracle.com/webfolder/technetwork/data-quality/edqhelp/index.htm). An understanding of the concepts involved in the EDQ matching process will greatly aid understanding of the material presented here.

EDQ 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 designed to match a customer data snapshot against a list of prohibited persons. The process contains a match processor that is configured to produce a list of customer reference numbers and related prohibited person identifiers. Each data stream that is input to, or output from, the match processor, is considered to be a sub-process of the match processor. Therefore, there are three sub-processes in this example, 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 2 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, using 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, queued and passed 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 EDQ 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 EDQ Web Application Server.

3.4 Real-Time Processing

EDQ is capable of processing messages in real time. Currently, EDQ supports messaging using:

- 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 may 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 and published to the EDQ Dashboard. The interval can be determined either by the number of records processed or by time limit. When an interval limit is reached, EDQ 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.

4 Application Security

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

4.1 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 EDQ. The Java Application Server can be configured to use either HTTP or HTTPS.

4.2 Authentication

EDQ authenticates user passwords against values held in the Config database or in a Lightweight Directory Access Protocol (LDAP) enabled user management server. The passwords are held in a hashed form that the application cannot reverse. This configuration is used by:

- the client user applications
- the EDQ web pages

The client user applications authenticate users using a proprietary protocol over HTTP or HTTPS. Passwords are encrypted before being sent to the server.

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

4.3 Storing Security Information

EDQ stores security information in a number of places, depending on the nature of the information:

Connection details for databases that EDQ connects to in order to perform snapshots and dynamic value lookups are stored in the Config schema. This includes user names, passwords, hosts names, and port numbers for the database connections. Passwords are stored in an encrypted form in the Config schema (or in a keystore on WebLogic platforms) and decrypted by EDQ when it needs to log into these databases. The decrypted password is not shown to EDQ users or administrators. The encryption/decryption key for the passwords is generated randomly for each installation of EDQ. On WebLogic platforms, the key is retained in the Key Store. On other platforms, it is stored in a file in the EDQ configuration directory. This random key generation on a per-installation basis ensures that encrypted passwords cannot be copied meaningfully between systems.

In WebLogic installations, and other installations where EDQ uses Java Naming and Directory Interface (JNDI) connections to connect to its repository databases, the authentication details to the database are stored in the Application Server. EDQ uses them by referencing the JNDI names in a file (`director.properties`) in the EDQ configuration directory (`oedq_home`).

4.4 Data Segmentation

When EDQ 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. EDQ 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.

5 Related Documents

For more information, see the following documents in the documentation set:

- *Oracle Enterprise Data Quality Installation Guide*

See the latest version of this and all documents in the Oracle Enterprise Data Quality Documentation website at:

`http://download.oracle.com/docs/cd/E48549_01/index.htm`

Also, see the latest version of the EDQ Online Help, bundled with EDQ.

6 Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Oracle Enterprise Data Quality Architecture Guide, Release 11g R1 (11.1.1.9)
E55995-01