Oracle® Fusion Middleware

Understanding Oracle Enterprise Data Quality

12*c* (12.2.1.4.0)

E95649-02

December 2022

Oracle Fusion Middleware Understanding Oracle Enterprise Data Quality, 12*c* (12.2.1.4.0)

E95649-02

Primary Author: Oracle Corporation

Contents

Preface

1 Overview of Oracle Enterprise Data Quality

2 Understanding Key Concepts of Enterprise Data Quality

Preface

An overview, the main terms, data processing, and basic steps to use are presented in this guide.

Audience

This guide is intended for anyone interested in an overview of the key concepts and architecture of .

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`.

Accessible Access to Oracle Support

Oracle customers who have purchased support 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.

Related Documents

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

EDQ Documentation Library

The following publications are provided to help you install and use EDQ:

-
-
-
-
-
-
-
-

Find the latest version of these guides and all of the Oracle product documentation at

`https://docs.oracle.com`

Online Help

Online help is provided for all user applications. It is accessed in each application by pressing the **F1** key or by clicking the Help icons. The main nodes in the Director project browser have integrated links to help pages. To access them, either select a node and then press **F1**, or right-click on an object in the Project Browser and then select **Help**. The EDQ processors in the Director Tool Palette have integrated help topics, as well. To access them, right-click on a processor on the canvas and then select **Processor Help**, or left-click on a processor on the canvas or tool palette and then press **F1**.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Overview of Oracle Enterprise Data Quality

This chapter gives an overview of Oracle Fusion Middleware and Oracle Enterprise Data Quality.
This chapter includes the following sections:

1.1 Oracle Fusion Middleware Overview

Oracle Fusion Middleware is a collection of standards-based software products that spans a range of tools and services: from Java EE and developer tools, to integration services, business intelligence, and collaboration. Oracle Fusion Middleware offers complete support for development, deployment, and management of applications. Oracle Fusion Middleware components are monitored at run time using Oracle Enterprise Manager Fusion Middleware Control Console.

1.2 About Oracle Enterprise Data Quality

EDQ provides a comprehensive data quality management environment that is used to understand, improve, protect and govern data quality. EDQ facilitates best practice master data management, data integration, business intelligence, and data migration initiatives. EDQ provides integrated data quality in customer relationship management and other applications.

Following are the key features of EDQ:

- Integrated data profiling, auditing, cleansing and matching
- Browser-based client access
- Ability to handle all types of data (for example, customer, product, asset, financial, and operational)
- Connection to any Java Database Connectivity (JDBC) compliant data sources and targets
- Multi-user project support (role-based access, issue tracking, process annotation, and version control)
- Services Oriented Architecture (SOA) support for designing processes that may be exposed to external applications as a service
- Designed to process large data volumes
- A single repository to hold data along with gathered statistics and project tracking information, with shared access
- Intuitive graphical user interface designed to help you solve real world information quality issues quickly

- Easy, data-led creation and extension of validation and transformation rules
- Fully extensible architecture allowing the insertion of any required custom processing

1.3 Understanding the Software Components

EDQ is a Java Web Application that uses a Java Servlet Engine, a Java Web Start graphical user interface, and a Structured Query Language (SQL) relational database management system (RDBMS) system for data storage.

EDQ is a client-server architecture. It is comprised of several client applications that are Graphical User Interfaces (GUIs), a data repository, and a business layer. This section provides details on the architecture of these components, their data storage, data access, and I/O requirements.

1.3.1 What Are the Client Applications?

Provides a number of client applications that are used to configure and operate the product. Most are Java Web Start applications, and the remainder are simple web pages. The following table lists all the client applications, how they are started, and what each does:

Application Name	Starts In	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 data quality key performance indicators 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 differences 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
Configuration Analysis	Web Start	Analyze project configurations, and report on differences'.

The client applications can be accessed from the Launchpad on the 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. 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.

Since it 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 to add a user application for screening data against watchlists.

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

1.3.1.1 Where Is Data Stored?

The client computer only stores user preferences for the presentation of the client applications, while all other information is stored on the EDQ server.

1.3.1.2 Network Communications

The client applications communicate over either an Hypertext Transfer Protocol (HTTP) or a Secure Hypertext Transfer Protocol (HTTPS) connection, as determined by the application configuration on start-up. For simplicity, this connection is referred to as 'the HTTP connection' in the remainder of this document.

1.3.2 How is Data Stored in the EDQ Repository?

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, then the configuration cannot be shared by pointing both servers to the same schemas.

1.3.2.1 What Is the 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.

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.

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

1.3.2.2 What Is the 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.

The amount of data held in the Results schema will 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.

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 application programming interfaces (APIs)
- Queried using full table scans to support process execution
- Indexed
- Queried using complex SQL statements in response to user interactions with the client applications
- 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.

1.3.3 Where does EDQ Store Working Data on Disk?

EDQ uses two configuration directories, which are separate from the installation directory that contains the program files. These directories are:

- The *base* configuration directory: This directory contains default configuration data. This directory is named `oedq.home` in an Oracle WebLogic installation but can be named anything in an Apache Tomcat installation.
- The *local* configuration directory: This directory contains overrides to the base configuration, such as data for extension packs or overrides to default settings. EDQ looks in this directory first, for any overrides, and then looks in the base directory if it does not find the data it needs in the local directory. The local configuration directory is named `oedq.local.home` in an Oracle WebLogic installation but can be named anything in an Apache Tomcat installation.

Some of the files in the configuration directories are used when processing data from and to file-based data stores. Other files are used to store server configuration properties, such as which functional packs are enabled, how EDQ connects to its repository databases, and other critical information.

The names and locations of the home and local home directories are important to know in the event that you need to perform any manual updates to templates or other individual components.

These directories are created when you install EDQ.

1.3.4 What Is the Business Layer?

The business layer fulfills three main functions:

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

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

When passing data to and from the client application, 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 client application 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 Admin Guide.

2

Understanding Key Concepts of Enterprise Data Quality

This chapter provides information about key EDQ concepts.
This chapter includes the following sections.

2.1 Understanding EDQ Terms

The most important terms used in EDQ are:

Project
A group of related processes working on a common set, or sets, of data using shared reference data.

Data Store
A connection to a store of data, whether the data is stored in a database or in one or more files. The data store may be used as the source of data for a process, or you may export the written Staged Data results of a process to a data store, or both.

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.

Processor
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.

Published Processors
Additional processors (in addition to those in the Processor Library) that have been installed from an Extension Pack (such as, the Customer Data pack) or published onto the server by EDQ users. They are included in the Project Browser so that they can be packaged like

other objects.There are three types of Published processors: Template, Reference, and Locked Reference

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
Consists of data snapshots and data written by processes and is stored within the Results schema.

Snapshot
A captured copy of external data stored within the EDQ repository.

Job
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.

Images
Customizing the icon associated to a processor.

Exports
There are two types of exports:

- A prepared export (Staged Data Export or Results Book Export) that uses a saved configuration.
- An ad-hoc export of the current results from the Results Browser to an Excel file.

Web Services
Used to deploy processes (as configured in Director) to provide easy integration with source systems for real time data auditing, cleansing, and matching (for example, for real time duplicate prevention).

Run Profiles
Optional templates that specify configuration settings that you can use to override the default job run settings.

Issues
Allows users to keep a record of their key findings when analyzing data, and also provides a way for work on a project to be allocated and tracked amongst several users.

Project Notes
Allows you to use Director as the definitive repository for all information associated with the project. Any information that needs to be made available to all users working on a project can be added as a note throughout the progress of a project.

2.2 What Is 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.

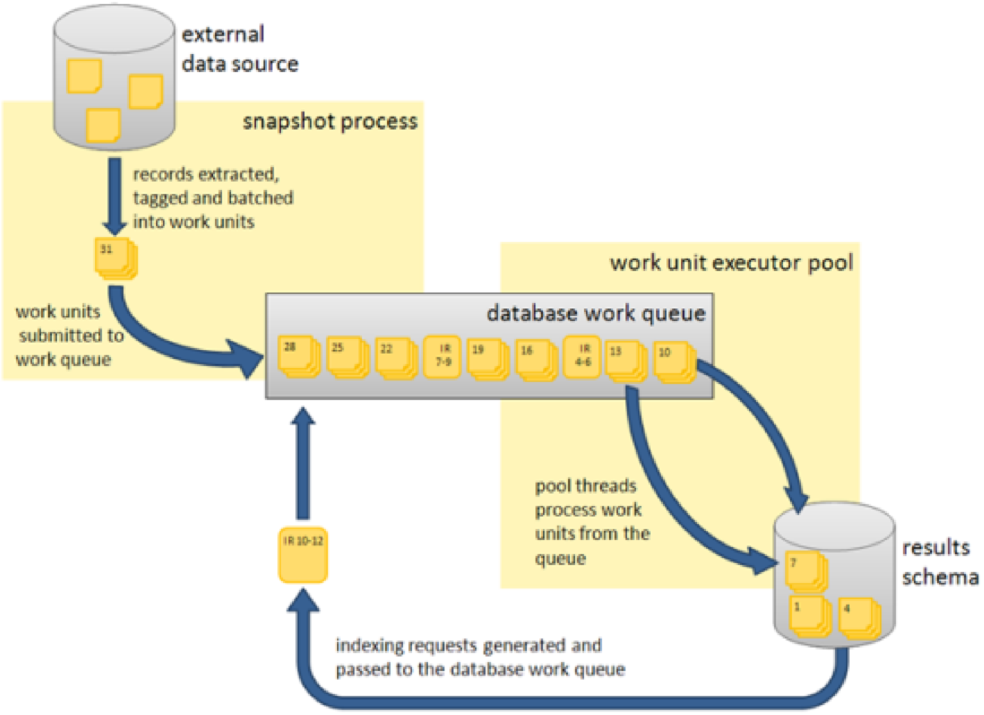
As the data is retrieved, it is processed by a single thread. This involves:

- 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 2-1   The Data Capture Process



2.2.1 Understanding 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.

2.3 What Is 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.

2.3.1 What Is 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.

2.3.2 What Is 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. 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 among 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.

2.3.3 What Is Whole Record Set 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.

2.3.4 What Are 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.

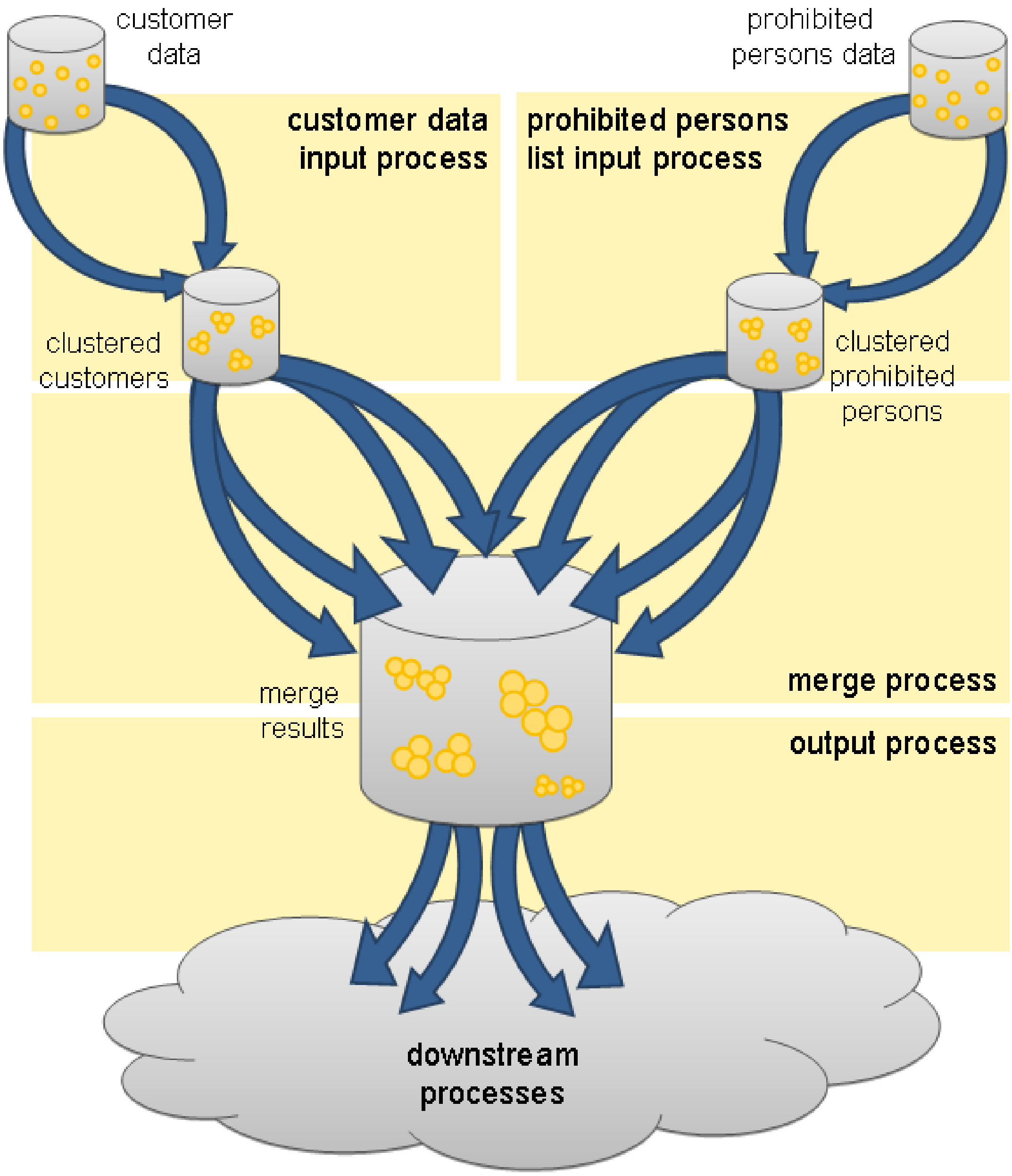
> **Note:**
>
> Jobs may be designed independently of any specific source or target of data. Such jobs will normally be run with a set of command line parameters, or a stored Run Profile that sets the same parameters, that dynamically change key configuration points such as the physical source of the 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.

2.4 What Is Match Processing?

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 2-2 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:

Phase	Description
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.

> **Tip:**
>
> Oracle recommends that the reader familiarizes themselves with the material contained in the Online Help regarding matching concepts..

2.5 What Is 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.

3

High Availability for EDQ

EDQ can be deployed in a number of ways to support High Availability requirements. The most common and recommended way, is to use WebLogic Clustering to support a highly available EDQ system. WebLogic Clustering allows multiple EDQ servers to share the same installation and configuration and these can be managed and operated as a single system with built-in redundancy. Simple High Availability of EDQ real-time services, however, can be achieved simply by configuring multiple servers with identical configuration, supporting real-time services and load balancing requests between them. In both cases, Oracle Real Application Clustering (RAC) is recommended to ensure high availability of the database tier, though it may be noted that most EDQ real-time services do not require database availability once they are running.

This chapter describes how EDQ operates when deployed to a WebLogic cluster and includes the following sections.

Technology Requirements

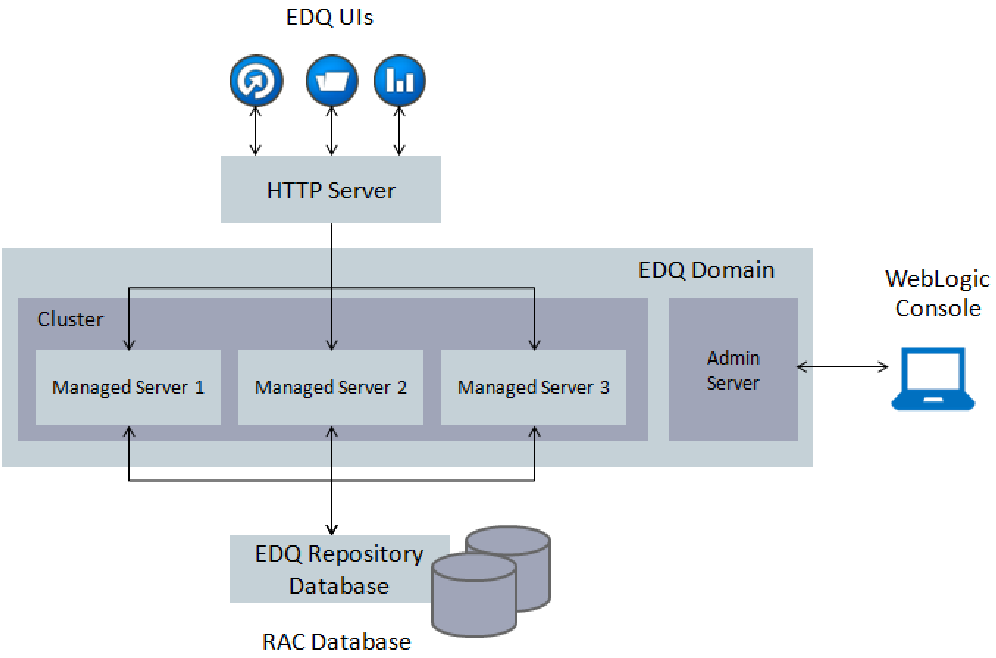
EDQ clustering in WebLogic is based on WebLogic and Coherence technology, and requires an appropriate license that allows WebLogic and Coherence clustering to be used. EDQ clustering in Tomcat uses Coherence Community Edition, which is open source. The use of Oracle Real Application Clustering (RAC) is recommended for High Availability of the Database and requires an appropriate license that allows RAC to be used.

3.2 EDQ in a WebLogic Cluster

EDQ fully supports operation in a WebLogic Cluster. For information on installing EDQ in a cluster, see the "Configuring Enterprise Data Quality with Oracle WebLogic Server" chapter in .

Figure 3-1 shows EDQ in a WebLogic Cluster:

Figure 3-1 EDQ in a WebLogic Cluster



3.2.1 Characteristics of EDQ in a WebLogic Cluster

When deployed in a cluster, EDQ exhibits the following general characteristics:

- All servers in the cluster share the same schemas in the repository database (EDQ Config and EDQ Results and EDQ Staging). Connections to those schemas are defined as JNDI Data Sources in WebLogic. These data sources may be configured for Oracle RAC to allow the use of multiple database nodes and to load balance connections, and this is recommended for High Availability.
- Stateless real-time jobs run on ALL managed servers in the cluster. See "Job Definitions When Running in a Cluster" for more information on how the system detects a stateless real-time job.
- A batch job runs on one of the managed servers in the cluster, normally the least busy server. For example, in a cluster of two servers, if server A is currently running a batch job and another is submitted, the new batch job will run on server B.
- Where Oracle RAC is used and one of the nodes is stopped or fails:
  - Running real-time jobs usually continue processing as normal.
  - Running batch jobs using connections to a failed node will fail, however, can be restarted as the newly submitted batch job will request new connections to an active node.
  - If the system is idle, there is no impact.
- Where Oracle RAC is not used and the database is not available:
  - Running real-time jobs usually continue processing as normal.

  - Running batch jobs fail and cannot be restarted successfully until the database is made available again.
  - If the system is idle, there is no impact.
  - The EDQ application server does not require a restart, provided the JNDI connection pool is unsuspended once the database is available again.
- When a managed server in the cluster fails, it can be detected by the load balancer and the request router (such as Oracle HTTP Server) no longer routes requests to this managed server. Batch jobs are processed by one of the live servers.

3.2.2 Job Definitions When Running in a Cluster

EDQ automatically detects if a job is a stateless real-time production job that can run on all servers in a cluster, or if it includes either batch processing or results writing, and therefore should run on a single server in the cluster.

To be detected as a real-time job that runs on all servers in a cluster, a job must have the following properties:

- It must have a single phase only.
- All process chains must be connected to a real-time reader or a Data Interface reader that is configured to use a real-time mapping.
- It must not write to staged data or reference data.

> **Note:**
>
> Jobs which only write to the landing area are not treated as writing staged data assuming they export data via a Data Interface. The job may contain such writers, as long as they are disabled when running the job, for example, in the job definition or Run Profile.

- It must not publish results to staged data.
- It must be run with a Run Label.

> **Note:**
>
> Jobs that use Interval mode to write results data, write results for review in Match Review, or publish to the Dashboard cannot run on all servers.

The following types of jobs are theoretically possible. These attempt to run on all servers but do not run successfully on all servers:

- A real-time job with a 'before' phase trigger that runs a batch job. This type of job should be reconfigured such that the real-time job is triggered at the end of the batch job.
- A real-time job that writes to reference data or staged data. This type of job runs successfully on one server only, but fails on other servers with an 'already locked' message. This type of job should not be used in production. It should be run on a single server by not running it with a run label so that results and written data can be reviewed. If there is a need to write out data from a production stateless real-time job that runs on

all servers, this should be done using a Writer that writes through a Data Interface to an Export running in Append mode.

- A real-time job that is run more than once at the same time. The first instance of the job runs on all servers, however, subsequent attempts will fail.

All other types of job, such as batch jobs and real-time jobs that write results or are without a run label, run on the least busy managed server in the cluster. This means that some types of jobs that were created in versions of EDQ before 12.2.1 and that are currently considered as real-time jobs will not be considered as jobs that can run on all servers. This is particularly true of jobs that have an initial phase that prepares data (for example, reference data for a real-time match process) in batch, and a later phase that performs real-time processing. If there is a requirement to run the real-time phase of such jobs on all servers, the job should be split into two jobs - a batch job that prepares the reference data and a real-time job that meets the requirements above. The batch job should have an end of final phase trigger that starts the real-time job. This will then mean that the batch job will run on a single server and the real-time job will run on all servers.

3.2.3 Monitoring EDQ Web Services

Most load balancers are aware of whether managed servers are running or not, but are not necessarily aware of whether this actually means that the application's services are 'ready'. EDQ real-time jobs supporting web services are normally intended to be running whenever the managed server is running, and therefore scheduled to start on server start-up. This can be achieved using an "Autorun" chore. A clustered environment provides advantages here. Since stateless real-time jobs run on all servers in a cluster, any running real-time jobs will automatically start up on a new managed server that is added to the cluster with no need for manual intervention.

In addition, EDQ integrates with the Fusion Middleware 'Ready App' feature, which allows a URL to be checked to determine if an application is ready to receive web service requests or not.

The URL to check for readiness is `[http://server:port/weblogic/ready]`, so for example it might be `[http://host234:8001/weblogic/ready]`for an EDQ managed server.

When a server is ready to service requests the URL returns a status of **200**. If the server is not ready it will return **503**. This then provides an easy way for load balancers to detect service availability.

When an EDQ server starts up, it assumes the use of Autorun or a schedule to immediately start the required real-time jobs. However, if those jobs are stopped or fail for any reason (and the managed server is still running), the EDQ server can declare itself 'not ready' by use of a trigger. EDQ's default implementation of the Ready App integration is set up to monitor the availability of all the Customer Data Services Pack real-time services, but can be modified to monitor the real-time services of your choice. It is implemented using an EDQ trigger in the EDQ 'home' configuration directory:

```
[domain home]/edq/config/fmwconfig/edq/oedq.home/triggers/readyappjobfail.groovy
```

To modify this, such that the failure of a different set of named jobs will result in an EDQ server declaring itself 'not ready', copy the file to the triggers directory in the `[oedq.local.home]` directory and modify the projects and missions lists to correspond with the EDQ projects and jobs that you want to monitor.

3.2.4 UI Behavior

If EDQ is deployed in a cluster, it is possible to use a front-end load balancer, such as Oracle HTTP Server, and load balance user sessions across the cluster. In this scenario, a user accessing the system using a load-balanced URL is connected to one of the managed servers, and all subsequent actions will be on that server. In most aspects, it is immaterial to users which server they are connected to, as the configuration and results databases are the same. Jobs and tasks that are active on the cluster are displayed to Director and Server Console users regardless of which of the managed servers they are connected to. The server that is running each job is displayed in the UI.

To customize the EDQ web pages, such as the Launchpad, to display the name of the managed server to which the user session is connected in the header, add the following line to `director.properties` in the EDQ Local Home directory:

```
[expr]adf.headerextra = ': ' || weblogic.Name
```

UI sessions, in both the WebStart applications and the EDQ Launchpad, are not automatically failed over. In the event of managed server failure, users connected to that managed server need to log in again. Provided at least one other managed server is available, the log in attempt should be successful as the user is then connected to an active managed server.

3.2.5 Tuning

As EDQ Clustering uses Coherence, it may be necessary to tune certain aspects of the system by adjusting Coherence parameters. For information, see the Tuning EDQ Performance section in *Administering Oracle Enterprise Data Quality* guide.

To avoid potential issues when restarting a previously failed server in a cluster that can cause client disconnection issues (to other managed servers), Oracle recommends setting the **Member Warmup Timeout** property of the Cluster to **30** (seconds) using the WebLogic Console (as opposed to the default value of 0). This setting is found under the **General** settings for the cluster.

3.2.6 Use of Oracle Web Services Manager

If there are other managed servers in the domain using the Oracle Web Services Manager Policy Manager ('wsm-pm'), then all EDQ managed servers need to be created referencing the `WSMPM-MAN-SVR` server startup group in the WebLogic Domain Configuration Wizard, in addition to the EDQ managed server startup group that is created by default.

3.2.7 Management Ports

If you are running EDQ as a single managed server and not using a WebLogic cluster, the default management port is 8090. If you are running EDQ in a WebLogic cluster, EDQ uses random ports which are allocated on server start. You can examine the Server Access MBean to find the current ports, but note that these will change when the EDQ servers are restarted.

The standard EDQ JMX client has been updated to query these server access beans automatically, using the MBean server in the Administration server. Specify the host name and port for the Administration server and, the connector code will redirect the request to one of the managed servers on the correct port.

```
$ java -jar jmxtools.jar runjob ... adminhost:7001
```

The username and password with jmxtools must be valid in WebLogic and EDQ. Use a user from the authentication provider setup in OPSS. If you are using a single EDQ server, there is no advantage in running this in a cluster. Remove the server from the cluster and, the default port 8090 will be used.

It is possible to configure different, fixed, management port for each server in a cluster. (If the servers are running on different machines, they can use the same port). Edit the `oedq.local.home` version of `director.properties` and replace the current management port setting with:

```
[expr]management.port = 8090 + servernum - 1
```

Here servernum is a precomputed value which is 1 for edq_server1, 2 for edq_server2 and so on. Using this example the management port for edq_server1 will be 8090, the port for edq_server2 will be 8091, so on.

The Siebel configuration in `dnd.properties` will be required to use the port for a specific managed server. If you are running EDQ as a single managed server and not using a WebLogic cluster, the default management port is 8090.

To discover the management (JMX) ports for any managed server, use Enterprise Manager Fusion Middleware Control as follows:

1. Click on the domain to bring up the drop-down list of options and select the **System MBean Browser**.
2. Navigate to **Application Defined Mbeans > edq > Server:** *servername* **> ServerAccess > ServerAccess**.

3.2.8 Assumptions

EDQ uses a number of files in the filing system of a given server. In most cases, for example for most of the files in the EDQ configuration directories, these files are accessed in a read-only fashion and do not change. It is therefore possible to pack and unpack the WebLogic domain onto several machines.

However, some types of EDQ jobs, especially batch jobs, require access to the EDQ file landing area in order to run. For correct operation of a clustered system, this file landing area should be mounted on shared file storage (for example, NFS) so that managed servers on different machines can access it. This then means that if a job is triggered by the presence of a file in the landing area, it can still run on any managed server on any machine, or if one EDQ job writes a file to the landing area that is then consumed by another job, both jobs could run on any machine as they have shared access to the landing area.

Note that a Best Practice for Fusion Middleware High Availability is to mount the whole domain on shared file storage, which ensures this requirement is met without the need to move the landing area specifically. For information, see *Oracle Best Practices for High Availability*.

If the landing area is specifically moved to shared storage, the operating system user account that runs the EDQ application server must have read and write access to it, and `director.properties` in the EDQ local home directory must be updated to add the following property to point to it:

```
landingarea=[path]
```

3.2.9 Limitations and Exclusions

As noted above, in normal operation and in most use cases, EDQ real-time jobs continue to work in the event of any disruption in repository database availability. This is true for all real-time jobs that are provided in the Customer Data Services Pack, and for most custom-configured real-time jobs.

However, the following limitations must be noted:

- If a real-time job performs Reference Data Matching using one or match processors with staged or reference data input into its Reference Data port, any match processors in the processes used must be configured to cache the prepared reference data into memory. This is enabled using a simple tick-box option in the Advanced Options of the match processor.
- If a real-time job publishes data to Case Management, it fails messages until the database is available again. The real-time job itself continues running.
- If a real-time job uses an External Lookup that directly looks up into an external database, rather than Staged Data or Reference Data, which is normally cached in memory, the lookup may fail and this may fail messages until the database being looked up is available again.
- Similarly, if a real-time job uses a large Reference Data set using Staged or Reference Data, but over the configurable cache limit, which cannot be cached in memory, the lookup will fail and this will fail any messages until the database is available again. Note that the default maximum row limit for caching a Reference Data set is 100,000 rows, but this can be modified by adding the following line in `director.properties` in the EDQ Local Home:

  ```
  resource.cache.maxrows = [rowlimit]
  ```

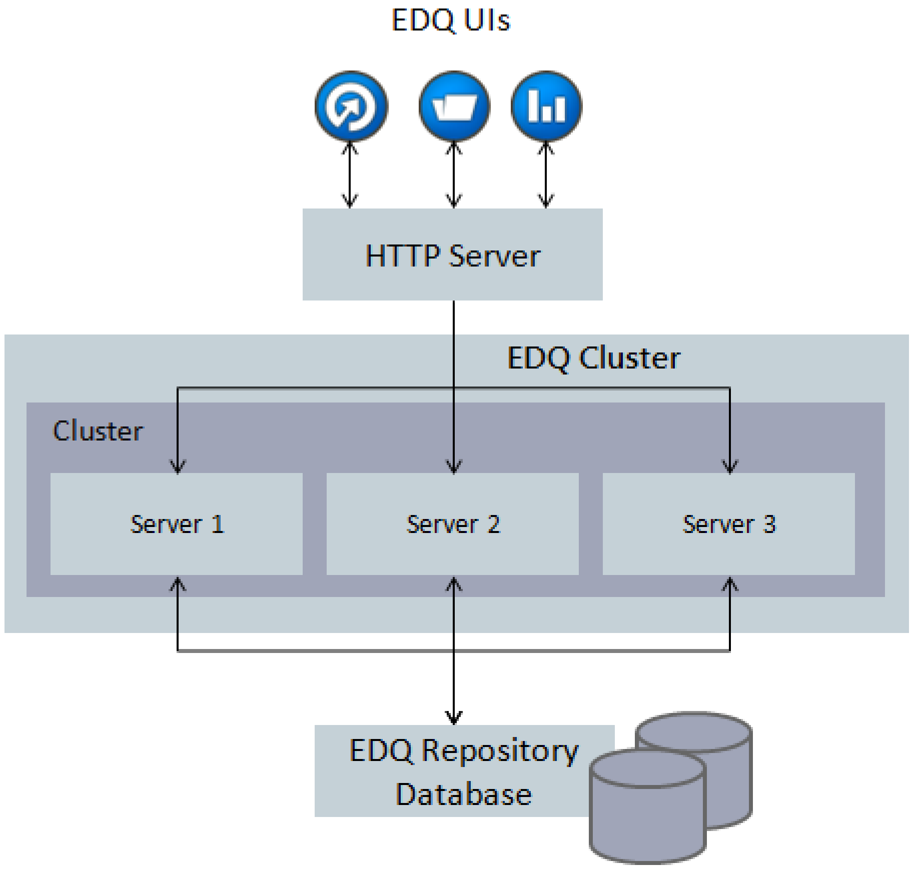
  where `[rowlimit]` is the maximum number of rows, for example 100000.
- If a real-time job uses a user account in an external realm, it is dependent on the LDAP server being contactable to lookup user details.

3.3 EDQ in a Tomcat Cluster

EDQ fully supports operation in a Tomcat Cluster. For information on installing EDQ in a cluster, see Configuring Enterprise Data Quality with Apache Tomcat.

Figure 3-2 shows EDQ in a Tomcat Cluster:

Figure 3-2 EDQ in a Tomcat Cluster



3.3.1 Configuring Coherence in Tomcat

EDQ clustering uses Coherence, which can be configured using environment variables and system properties. You can use an operational override XML file for more complex cases. This topic covers cluster discovery and setup using unicast addressing. Multicast can also be used but the configuration is generally more complex.

Set up environment variables

Set the following environment variables for all the Apache Tomcat processes in the cluster:

- **COHERENCE_CLUSTER**: The name for this cluster. Normally all servers will be in the same cluster, but it is possible to have multiple clusters running on the same servers with different names.
- **COHERENCE_WKA**: The host or Kubernetes service name for the system that will always have an EDQ instance running. You can specify multiple well-known-addresses using an override file.

Configure system properties

Set the system properties **server.name** and **coherence.member** for each Tomcat process to the name of the server in the cluster. The server name can be a host name or a simple name such as `server1` or `server2`.

Edit the server.xml file

Use the **jvmRoute** property if you use a web server front end with the AJP protocol. To add **jvmRoute** edit the Tomcat *server.xml* as follows:

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="${server.name}">
```

> **Note:**
>
> In Tomcat versions 10.0 and earlier you can set **jvmRoute** using a system property.

If you want to share the *server.xml* configuration amongst instances running on the same server, you can set the ports using system properties:

```
<Server port="${control.port}" shutdown="SHUTDOWN">
   ...
    <Connector port="${http.port}" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
    ...
    <Connector protocol="AJP/1.3"
               address="::1"
               port="${ajp.port}"
               redirectPort="8443" />
    ...
    <Engine name="Catalina" defaultHost="localhost" jvmRoute="$
{server.name}">
    ...
</Server>
```

Setup an operational override file

To setup an operational override file, create a file `tangosol-coherence-override.xml` to include the cluster configuration. For example, to set the member name to the **system.name** property and to specify multiple well-known-addresses include the following in the file:

```
<?xml version='1.0'?>
<coherence  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns="http://xmlns.oracle.com/coherence/coherence-operational-
config"
            xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-
operational-config coherence-operational-config.xsd">

  <cluster-config>

    <member-identity>
      <member-name system-property="server.name"></member-name>
    </member-identity>
    <unicast-listener>
      <well-known-addresses>
        <address>host1</address>
        <address>host2</address>
        ...
      </well-known-addresses>
    </unicast-listener>

  </cluster-config>
</coherence>
```

Edit the **address** elements to list all the servers that will run cluster members. You can place the file in the WEB-INF/classes directory in the expanded EDQ war file, or add it to the Tomcat classpath.

To add the override file to the Tomcat classpath, save the `tangosol-coherence-override.xml` file to a directory and add this location to the **common.loader** setting in the Tomcat *catalina.properties* file:

```
common.loader="${catalina.base}/lib","${catalina.base}/lib/*.jar","$
{catalina.home}/lib","${catalina.home}/lib/*.jar",\
        coherenceconfigdirectory
```

Here, replace `coherenceconfigdirectory` with the path to the directory that contains the override XML file.

Configure EDQ

To configure EDQ, add the property `cluster.mode = true` to *director.properties*.

3.3.2 Characteristics of EDQ in a Tomcat Cluster

When deployed in a Tomcat cluster, EDQ exhibits the same general characteristics as those when deployed in a Weblogic cluster. See Characteristics of EDQ in a WebLogic Cluster.

3.3.3 Job Definitions When Running in a Cluster

EDQ automatically detects if a job is a stateless real-time production job that can run on all servers in a cluster, or if it includes either batch processing or results writing, and therefore should run on a single server in the cluster. Job definitions when running in a Tomcat cluster are the same as for a Weblogic cluster. See Job Definitions When Running in a Cluster for details.