Oracle® Fusion Middleware

Understanding Oracle Data Integrator

12*c* (12.2.1.2.0)

E77136-01

September 2016

ORACLE®

Oracle Fusion Middleware Understanding Oracle Data Integrator, 12*c* (12.2.1.2.0)

E77136-01

Primary Authors: Laura Hofman Miquel, Joshua Stanley, Aslam Khan

Contributing Authors: Alex Kotopoulis, Michael Reiche, Jon Patt, Alex Prazma, Gail Risdal

Contributors: David Allan, Linda Bittarelli, Sophia Chen, Pratima Chennupati, Victor Chow, Sowmya Dhandapani, Daniel Gallagher, Gary Hostetler, Kevin Hwang, Aslam Khan, Sebu T. Koleth, Christian Kurz, Venkata Lakkaraju, Thomas Lau, Deekshit Mantampady, Kevin McDonough, Luda Mogilevich, Ale Paez, Suresh Pendap, Sandrine Riley, Julien Testut, Sachin Thatte, Julie Timmons, Jay Turner, Vikas Varma, Robert Velisar, Winnie Wan, Geoff Watters, Jennifer Waywell

Contents

4 Managing Environments

5 Life Cycle Management in Oracle Data Integrator

Preface

This manual describes how to develop data integration projects using Oracle Data Integrator.

This preface contains the following topics:.

Audience

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

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 that 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 *Oracle Data Integrator Library*.

- Release Notes for Oracle Data Integrator
- Administering Oracle Data Integrator
- Developing Integration Projects with Oracle Data Integrator
- Installing and Configuring Oracle Data Integrator
- Upgrading Oracle Data Integrator
- Application Adapters Guide for Oracle Data Integrator

- Developing Knowledge Modules with Oracle Data Integrator
- Connectivity and Knowledge Modules Guide for Oracle Data Integrator
- Migrating From Oracle Warehouse Builder to Oracle Data Integrator
- Oracle Data Integrator Tool Reference
- Data Services Java API Reference for Oracle Data Integrator
- Open Tools Java API Reference for Oracle Data Integrator
- Getting Started with SAP ABAP BW Adapter for Oracle Data Integrator
- Java API Reference for Oracle Data Integrator
- Getting Started with SAP ABAP ERP Adapter for Oracle Data Integrator
- *Oracle Data Integrator 12c Online Help*, which is available in ODI Studio through the JDeveloper Help Center when you press **F1** or from the main menu by selecting **Help**, and then **Search** or **Table of Contents**.

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 Data Integrator

This chapter introduces Oracle Data Integrator.

This chapter includes the following sections:

Introduction to Data Integration with Oracle Data Integrator

Data Integration ensures that information is timely, accurate, and consistent across complex systems. This section provides an introduction to data integration and describes how Oracle Data Integrator provides support for Data Integration.

What is Data Integration?

Integrating data and applications throughout the enterprise, and presenting them in a unified view is a complex proposition. Not only are there broad disparities in technologies, data structures, and application functionality, but there are also fundamental differences in integration architectures. Some integration needs are Data Oriented, especially those involving large data volumes. Other integration projects lend themselves to an Event Driven Architecture (EDA) or a Service Oriented Architecture (SOA), for asynchronous or synchronous integration.

Data Integration ensures that information is timely, accurate, and consistent across complex systems. Although it is still frequently referred as Extract-Transform-Load (ETL), data integration was initially considered as the architecture used for loading Enterprise Data Warehouse systems. Data integration now includes data movement, data synchronization, data quality, data management, and data services.

About Oracle Data Integrator

Oracle Data Integrator provides a fully unified solution for building, deploying, and managing complex data warehouses or as part of data-centric architectures in a SOA or business intelligence environment. In addition, it combines all the elements of data integration—data movement, data synchronization, data quality, data management, and data services—to ensure that information is timely, accurate, and consistent across complex systems.

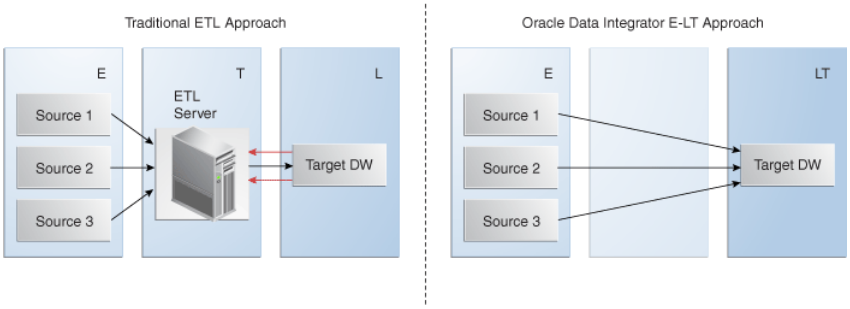
Oracle Data Integrator (ODI) features an active integration platform that includes all styles of data integration: data-based, event-based and service-based. ODI unifies silos of integration by transforming large volumes of data efficiently, processing events in real time through its advanced Changed Data Capture (CDC) framework, and providing data services to the Oracle SOA Suite. It also provides robust data integrity

control features, assuring the consistency and correctness of data. With powerful core differentiators - heterogeneous E-LT, Declarative Design and Knowledge Modules - Oracle Data Integrator meets the performance, flexibility, productivity, modularity and hot-pluggability requirements of an integration platform.

What is E-LT?

Traditional ETL tools operate by first *E*xtracting the data from various sources, *T*ransforming the data in a proprietary, middle-tier ETL engine that is used as the staging area, and then *L*oading the transformed data into the target data warehouse, integration server, or Hadoop cluster. Hence the term *ETL* represents both the names and the order of the operations performed, as shown in Figure 1–1.

Figure 1–1 Traditional ETL versus ODI E-LT



The data transformation step of the ETL process is by far the most compute-intensive, and is performed entirely by the proprietary ETL engine on a dedicated server. The ETL engine performs data transformations (and sometimes data quality checks) on a row-by-row basis, and hence, can easily become the bottleneck in the overall process. In addition, the data must be moved over the network twice – once between the sources and the ETL server, and again between the ETL server and the target data warehouse or Hadoop cluster. Moreover, if one wants to ensure referential integrity by comparing data flow references against values from the target data warehouse, the referenced data must be downloaded from the target to the engine, thus further increasing network traffic, download time, and leading to additional performance issues.

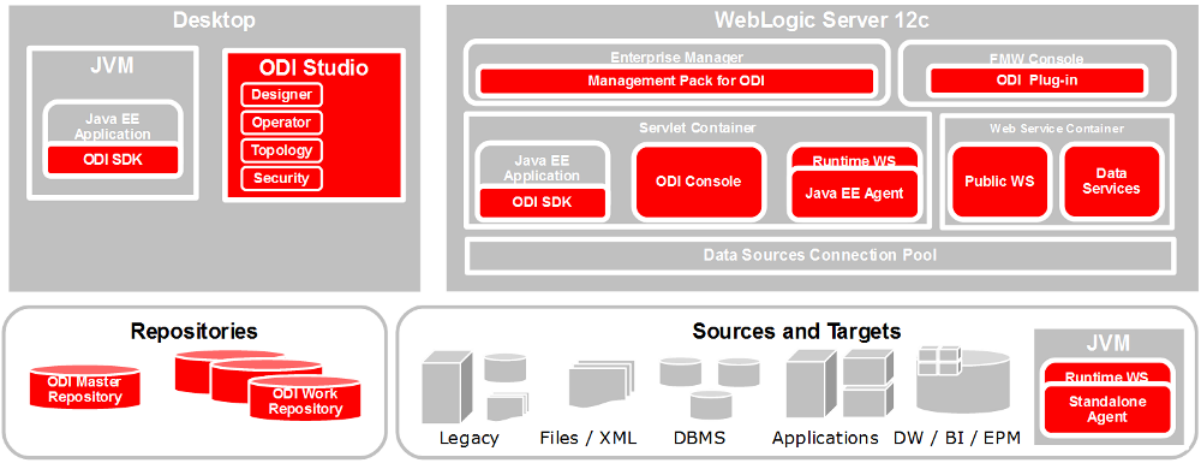
In response to the issues raised by ETL architectures, a new architecture has emerged, which in many ways incorporates the best aspects of manual coding and automated code-generation approaches. Known as *E-LT*, this new approach changes where and how data transformation takes place, and leverages existing developer skills, RDBMS and Big Data engines, and server hardware to the greatest extent possible. In essence, E-LT moves the data transformation step to the target RDBMS, changing the order of operations to: Extract the data from the source tables, Load the tables into the destination server, and then Transform the data on the target RDBMS using native SQL operators. Note, with E-LT there is no need for a middle-tier engine or server as shown in Figure 1–1.

Oracle Data Integrator supports both ETL- and E-LT-Style data integration. See "*Designing E-LT and ETL-Style Mappings*" in *Developing Integration Projects with Oracle Data Integrator* for more information.

Understanding the Oracle Data Integrator Component Architecture

The Oracle Data Integrator platform integrates in the broader Fusion Middleware platform and becomes a key component of this stack. Oracle Data Integrator provides its run-time components as Java EE applications, enhanced to fully leverage the capabilities of the Oracle WebLogic Application Server. Oracle Data Integrator components include exclusive features for Enterprise-Scale Deployments, high availability, scalability, and hardened security. Figure 1–2 shows the ODI component architecture.

Figure 1–2 Oracle Data Integrator Component Architecture



Repositories

The central component of the architecture is the Oracle Data Integrator Repository. It stores configuration information about the IT infrastructure, metadata of all applications, projects, scenarios, and the execution logs. Many instances of the repository can coexist in the IT infrastructure. The architecture of the repository is designed to allow several separated environments that exchange metadata and scenarios (for example: Development, Test, Maintenance and Production environments). In the figure above, two repositories are represented: one for the development environment, and another one for the production environment. The repository also acts as a version control system where objects are archived and assigned a version number. The Oracle Data Integrator Repository can be installed on an OLTP relational database.

The Oracle Data Integrator Repository is composed of a master repository and several Work Repositories. Objects developed or configured through the users are stored in one of these repository types.

There is usually only one master repository that stores the following information:

- Security information including users, profiles and rights for the ODI platform.
- Topology information including technologies, server definitions, schemas, contexts, languages, etc.
- Versioned and archived objects.

The Work Repository is the one that contains actual developed objects. Several work repositories may coexist in the same ODI installation (for example, to have separate

environments or to match a particular versioning life cycle). A Work Repository stores information for:

- Models, including schema definition, datastores structures and metadata, fields and attributes definitions, data quality constraints, cross references, data lineage etc.
- Projects, including business rules, packages, procedures, folders, Knowledge Modules, variables etc.
- Scenario execution, including scenarios, scheduling information and logs.

When the Work Repository contains only the execution information (typically for production purposes), it is then called an Execution Repository.

For more information on how to manage ODI repositories, see "*Administering Repositories*" in *Administering Oracle Data Integrator*.

Users

Administrators, Developers and Operators use the Oracle Data Integrator Studio to access the repositories. This Fusion Client Platform (FCP) based UI is used for administering the infrastructure (security and topology), reverse-engineering the metadata, developing projects, scheduling, operating and monitoring executions.

Business users (as well as developers, administrators and operators), can have read access to the repository, perform topology configuration and production operations through a web based UI called *Oracle Data Integrator Console*. This Web application can be deployed in a Java EE application server such as Oracle WebLogic.

ODI Studio provides four Navigators for managing the different aspects and steps of an ODI integration project:

- Topology Navigator
- Designer Navigator
- Operator Navigator
- Security Navigator

Topology Navigator

Topology Navigator is used to manage the data describing the information system's physical and logical architecture. Through Topology Navigator you can manage the topology of your information system, the technologies and their datatypes, the data servers linked to these technologies and the schemas they contain, the contexts, the language and the agents, as well as the repositories. The site, machine, and data server descriptions will enable Oracle Data Integrator to execute the same mappings in different environments.

Designer Navigator

Designer Navigator is used to design data integrity checks and to build transformations such as for example:

- Automatic reverse-engineering of existing applications or databases
- Graphical development and maintenance of transformations and mappings
- Visualization of data flows in the mappings
- Automatic documentation generation
- Customization of the generated code

The main objects you handle through Designer Navigator are Models and Projects.

Operator Navigator

Operator Navigator is the production management and monitoring tool. It is designed for IT production operators. Through Operator Navigator, you can manage your executions in the sessions, as well as the scenarios in production.

Security Navigator

Security Navigator is the tool for managing the security information in Oracle Data Integrator. Through Security Navigator you can create users and profiles and assign user rights for methods (edit, delete, etc) on generic objects (data server, datatypes, etc), and fine-tune these rights on the object instances (Server 1, Server 2, etc).

Run-Time Agent

At design time, developers generate scenarios from the business rules that they have designed. The code of these scenarios is then retrieved from the repository by the Run-Time Agent. This agent then connects to the data servers and orchestrates the code execution on these servers. It retrieves the return codes and messages for the execution, as well as additional logging information – such as the number of processed records, execution time etc. - in the Repository.

The Agent comes in three different flavors:

- **Standalone Agent**

  The standalone agent runs in a separate Java Virtual Machine (JVM) process. It connects to the work repository and to the source and target data servers via JDBC. Standalone agents can be installed on any server with a Java Virtual Machine installed. This type of agent is more appropriate when you need to use a resource that is local to one of your data servers (for example, the file system or a loader utility installed with the database instance), and you do not want to install a Java EE application server on this machine.

- **Standalone Colocated Agent**

  A standalone colocated agent runs in a separate Java Virtual Machine (JVM) process but is part of a WebLogic Server domain and controlled by a WebLogic Administration Server. Standalone colocated agents can be installed on any server with a Java Virtual Machine installed, but require a connection to the WebLogic Administration Server. This type of agent is more appropriate when you need to use a resource that is local to one of your data servers but you want to centralize management of all applications in an enterprise application server.

- **Java EE Agent**

  A Java EE agent is deployed as a web application in a Java EE application server, for example, Oracle WebLogic Server. The Java EE agent can benefit from all the features of the application server (for example, JDBC data sources or clustering for Oracle WebLogic Server). This type of agent is more appropriate when there is a need for centralizing the deployment and management of all applications in an enterprise application server, or when you have requirements for high availability.

These agents are multi-threaded java programs that support load balancing and can be distributed across the information system. This agent holds its own execution schedule which can be defined in Oracle Data Integrator, and can also be called from an external scheduler. It can also be invoked from a Java API or a web service. See "*Setting Up a Topology*" in *Administering Oracle Data Integrator* more information on how to create and manage agents.

Oracle Data Integrator Console

Business users (as well as developers, administrators and operators), can have read access to the repository, perform topology configuration and production operations through a web based UI called *Oracle Data Integrator Console*. This web application can be deployed in a Java EE application server such as Oracle WebLogic.

To manage and monitor the Java EE and Standalone Agents as well as the ODI Console, Oracle Data Integrator provides a plug-in that integrates with Oracle Enterprise Manager Cloud Control as well as Oracle Fusion Middleware Control Console.

ODI Domains

An ODI domain contains the Oracle Data Integrator components that can be managed using Oracle Enterprise Manager Cloud Control (EMCC). An ODI domain contains:

- Several Oracle Data Integrator Console applications. An Oracle Data Integrator Console application is used to browse master and work repositories.
- Several Run-Time Agents attached to the Master Repositories. These agents must be declared in the Master Repositories to appear in the domain. These agents may be Standalone Agents or Java EE Agents. See "*Setting Up a Topology*" in *Administering Oracle Data Integrator* for information about how to declare Agents in the Master Repositories.

In EMCC, the Master Repositories and Agent pages display both application metrics and information about the Master and Work Repositories. You can also navigate to Oracle Data Integrator Console from these pages, for example to view the details of a session. In order to browse Oracle Data Integrator Console in EMCC, the connections to the Work and Master repositories must be declared in Oracle Data Integrator Console. See *Installing and Configuring Oracle Data Integrator* for more information.

Materialized Client Libraries for SDK APIs

The materialized client libraries enable usage of SDK APIs from a non Oracle Home Environment.

The following three materialized client libraries are located in <OH>/odi/modules/clients directory:

- `oracle.odi.common.clientLib.jar`
- `oracle.odi.tp.clientLib.jar`
- `oracle.odi.sdk.clientLib.jar`

> **Note:** These libraries are available only in Enterprise installation.

To use SDK APIs from a non Oracle Home Environment, include the following in the classpath:

- All three materialized client libraries mentioned above.
- JDBC driver for the database on which the ODI Repository is installed.

After the libraries and JDBC driver are included in the classpath, you can create an `OdiInstance` and invoke the SDK APIs.

See also, see *Java API Reference for Oracle Data Integrator*.

2

Understanding Oracle Data Integrator Concepts

This chapter provides an introduction to the main concepts of Oracle Data Integrator.

This chapter includes the following topics:

Introduction to Declarative Design

To design an integration process with conventional ETL systems, a developer needs to design each step of the process: Consider, for example, a common case in which sales figures must be summed over time for different customer age groups. The sales data comes from a sales management database, and age groups are described in an age distribution file. In order to combine these sources then insert and update appropriate records in the customer statistics systems, you must design each step, which includes:

1. Load the customer sales data in the engine
2. Load the age distribution file in the engine
3. Perform a lookup between the customer sales data and the age distribution data
4. Aggregate the customer sales grouped by age distribution
5. Load the target sales statistics data into the engine
6. Determine what needs to be inserted or updated by comparing aggregated information with the data from the statistics system
7. Insert new records into the target
8. Update existing records into the target

This method requires specialized skills, depending on the steps that need to be designed. It also requires significant efforts in development, because even repetitive succession of tasks, such as managing inserts/updates in a target, need to be developed into each task. Finally, with this method, maintenance requires significant effort. Changing the integration process requires a clear understanding of what the process does as well as the knowledge of how it is done. With the conventional ETL method of design, the logical and technical aspects of the integration are intertwined. Declarative Design is a design method that focuses on "What" to do (the Declarative Rules) rather than "How" to do it (the Process). In our example, "What" the process does is:

- Relate the customer age from the sales application to the age groups from the statistical file
- Aggregate customer sales by age groups to load sales statistics

"How" this is done, that is the underlying technical aspects or technical strategies for performing this integration task – such as creating temporary data structures or calling loaders – is clearly separated from the declarative rules.

Declarative Design in Oracle Data Integrator uses the well known relational paradigm to declare in the form of a mapping the declarative rules for a data integration task, which includes designation of sources, targets, and transformations.

Declarative rules often apply to metadata to transform data and are usually described in natural language by business users. In a typical data integration project (such as a Data Warehouse project), these rules are defined during the specification phase in documents written by business analysts in conjunction with project managers. They can very often be implemented using SQL expressions, provided that the metadata they refer to is known and qualified in a metadata repository.

Declarative rules are implemented using Mappings, which connect sources to targets through a flow of components such as Join, Filter, Aggregate, Set, Split, and so on.

Table 2–1 gives examples of declarative rules.

Table 2–1 Examples of declarative rules

Declarative Rule	Type	SQL Expression
Sum of all amounts or items sold during October 2005 multiplied by the item price	Aggregate	`SUM( CASE WHEN SALES.YEARMONTH=200510 THEN SALES.AMOUNT*PRODUCT.ITEM_PRICE ELSE 0 END )`
Products that start with 'CPU' and that belong to the hardware category	Filter	`Upper(PRODUCT.PRODUCT_NAME)like 'CPU%' And PRODUCT.CATEGORY = 'HARDWARE'`
Customers with their orders and order lines	Join	`CUSTOMER.CUSTOMER_ID = ORDER.CUSTOMER_ID And ORDER.ORDER_ID = ORDER_LINE.ORDER_ID`
Reject duplicate customer names	Unique Key Constraint	`Unique key (CUSTOMER_NAME)`
Reject orders with a link to an non-existent customer	Reference Constraint	`Foreign key on ORDERS(CUSTOMER_ID) references CUSTOMER(CUSTOMER_ID)`

Introduction to Knowledge Modules

Knowledge Modules (KM) implement "how" the integration processes occur. Each Knowledge Module type refers to a specific integration task:

- Reverse-engineering metadata from the heterogeneous systems for Oracle Data Integrator (RKM). See "*Creating and Using Data Models and Datastores*" in *Developing Integration Projects with Oracle Data Integrator* for more information on how to use the RKM.

- Handling Changed Data Capture (CDC) on a given system (JKM). See "*Using Journalizing*" in *Developing Integration Projects with Oracle Data Integrator* for more information on how to use the Journalizing Knowledge Modules.
- Loading data from one system to another, using system-optimized methods (LKM). These KMs are used in mappings. See "*Creating and Using Mappings*" in *Developing Integration Projects with Oracle Data Integrator* for more information on how to use the Loading Knowledge Modules.
- Integrating data in a target system, using specific strategies (insert/update, slowly changing dimensions) (IKM). These KMs are used in mappings. See "*Creating and Using Mappings*" in *Developing Integration Projects with Oracle Data Integrator* for more information on how to use the Integration Knowledge Modules.
- Controlling Data Integrity on the data flow (CKM). These KMs are used in data model's static check and mappings flow checks. See "*Creating and Using Data Models and Datastores*" and "*Creating and Using Mappings*" in *Developing Integration Projects with Oracle Data Integrator* for more information on how to use the Check Knowledge Modules.
- Exposing data in the form of web services (SKM). See "*Creating and Using Data Services*" in *Administering Oracle Data Integrator* for more information on how to use the Service Knowledge Modules.

A Knowledge Module is a code template for a given integration task. This code is independent of the Declarative Rules that need to be processed. At design-time, a developer creates the Declarative Rules describing integration processes. These Declarative Rules are merged with the Knowledge Module to generate code ready for runtime. At runtime, Oracle Data Integrator sends this code for execution to the source and target systems it leverages in the E-LT architecture for running the process.

Knowledge Modules cover a wide range of technologies and techniques. Knowledge Modules provide additional flexibility by giving users access to the most-appropriate or finely tuned solution for a specific task in a given situation. For example, to transfer data from one DBMS to another, a developer can use any of several methods depending on the situation:

- The DBMS loaders (Oracle's SQL*Loader, Microsoft SQL Server's BCP, Teradata TPump) can dump data from the source engine to a file then load this file to the target engine
- The database link features (Oracle Database Links, Microsoft SQL Server's Linked Servers) can transfer data directly between servers

These technical strategies amongst others correspond to Knowledge Modules tuned to exploit native capabilities of given platforms.

Knowledge modules are also fully extensible. Their code is open and can be edited through a graphical user interface by technical experts willing to implement new integration methods or best practices (for example, for higher performance or to comply with regulations and corporate standards). Without having the skill of the technical experts, developers can use these custom Knowledge Modules in the integration processes.

For more information on Knowledge Modules, see the *Connectivity and Modules Guide for Oracle Data Integrator* and the *Knowledge Module Developer's Guide for Oracle Data Integrator*.

Introduction to Mappings

A mapping connects sources to targets through a flow of components such as Join, Filter, Aggregate, Set, Split, and so on.

A mapping also references the Knowledge Modules (code templates) that will be used to generate the integration process.

Datastores

A datastore is a data structure that can be used as a source or a target in a mapping. It can be:

- a table stored in a relational database
- a Hive table in a Hadoop cluster
- an ASCII or EBCDIC file (delimited, or fixed length)
- a node from a XML file
- a JMS topic or queue from a Message Oriented Middleware
- a node from a enterprise directory
- an API that returns data in the form of an array of records

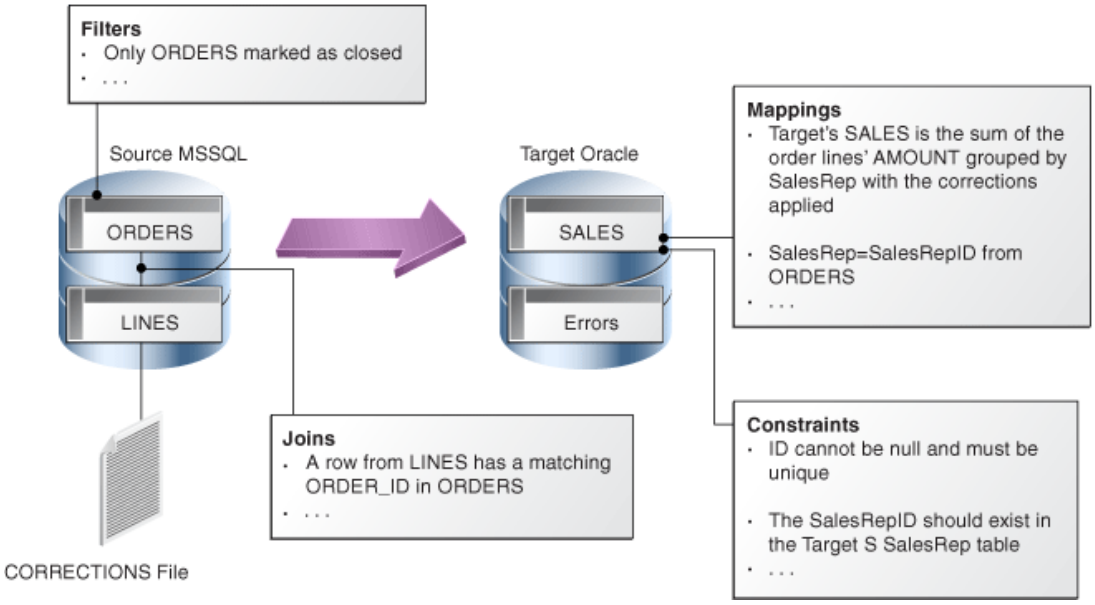
Regardless of the underlying technology, all data sources appear in Oracle Data Integrator in the form of datastores that can be manipulated and integrated in the same way. The datastores are grouped into data models. These models contain all the declarative rules –metadata - attached to datastores such as constraints.

Declarative Rules

The declarative rules that make up a mapping can be expressed in human language, as shown in the following example: Data is coming from two Microsoft SQL Server tables (ORDERS joined to LINES) and is combined with data from the CORRECTIONS file. The target SALES Oracle table must match some constraints such as the uniqueness of the ID column and valid reference to the SALES_REP table.

Data must be transformed and aggregated according to some mappings expressed in human language as shown in Figure 2–1.

Figure 2–1 Example of a business problem



Translating these business rules from natural language to SQL expressions is usually straightforward. In our example, the rules that appear in Figure 2–1 could be translated as shown in Table 2–2.

Table 2–2 Business rules translated

Type	Rule	SQL Expression/Constraint
Filter	Only ORDERS marked as closed	`ORDERS.STATUS = 'CLOSED'`
Join	A row from LINES has a matching ORDER_ID in ORDERS	`ORDERS.ORDER_ID = LINES.ORDER_ID`
Aggregate	Target's SALES is the sum of the order lines' AMOUNT grouped by sales rep, with the corrections applied	`SUM(LINES.AMOUNT + CORRECTIONS.VALUE)`
Mapping	Sales Rep = Sales Rep ID from ORDERS	`ORDERS.SALES_REP_ID`
Constraint	ID must not be null	ID is set to `not null` in the data model
Constraint	ID must be unique	A unique key is added to the data model with (ID) as set of columns
Constraint	The Sales Rep ID should exist in the Target SalesRep table	A reference (foreign key) is added in the data model on `SALES.SALES_REP = SALES_REP.SALES_REP_ID`

Implementing this business problem using Oracle Data Integrator is a very easy and straightforward exercise. It is done by simply translating the business rules into a mapping.

Data Flow

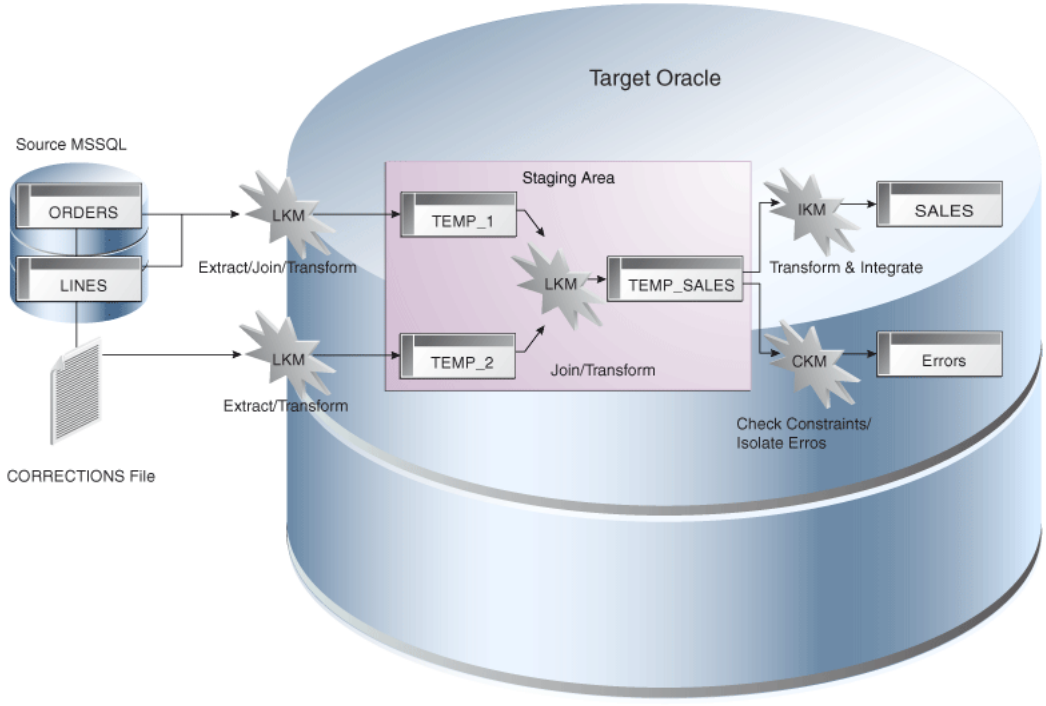
Business rules defined in the mapping are automatically converted into a data flow that will carry out the joins, filters, mappings, and constraints from source data to target tables.

By default, Oracle Data Integrator will use the Target RDBMS as a staging area for loading source data into temporary tables and applying all the required mappings, staging filters, joins and constraints. The staging area is a separate area in the RDBMS (a user/database) where Oracle Data Integrator creates its temporary objects and executes some of the rules (mapping, joins, final filters, aggregations etc.). When performing the operations this way, Oracle Data Integrator uses an E-LT strategy as it first extracts and loads the temporary tables and then finishes the transformations in the target RDBMS.

In some particular cases, when source volumes are small (less than 500,000 records), this staging area can be located in memory in Oracle Data Integrator's in-memory relational database – In-Memory Engine. Oracle Data Integrator would then behave like a traditional ETL tool.

Figure 2–2 shows the data flow automatically generated by Oracle Data Integrator to load the final SALES table. The business rules will be transformed into code by the Knowledge Modules (KM). The code produced will generate several steps. Some of these steps will extract and load the data from the sources to the staging area (Loading Knowledge Modules - LKM). Others will transform and integrate the data from the staging area to the target table (Integration Knowledge Module - IKM). To ensure data quality, the Check Knowledge Module (CKM) will apply the user defined constraints to the staging data to isolate erroneous records in the Errors table.

Figure 2–2 Oracle Data Integrator Knowledge Modules in action



Oracle Data Integrator Knowledge Modules contain the actual code that will be executed by the various servers of the infrastructure. Some of the code contained in the Knowledge Modules is generic. It makes calls to the Oracle Data Integrator Substitution API that will be bound at run-time to the business-rules and generates the final code that will be executed.

At design time, declarative rules are defined in the mappings and Knowledge Modules are only selected and configured.

At run-time, code is generated and every Oracle Data Integrator API call in the Knowledge Modules (enclosed by <% and %>) is replaced with its corresponding object name or expression, with respect to the metadata provided in the Repository. The generated code is orchestrated by Oracle Data Integrator run-time component - the Agent – on the source and target systems to make them perform the processing, as defined in the E-LT approach.

See "*Creating and Using Mappings*" in *Developing Integration Projects with Oracle Data Integrator* for more information on how to work with mappings.

3

Typical Integration Projects

Oracle Data Integrator provides a wide range of integration features. This chapter introduces the most typical ODI Integration Projects.

This chapter includes the following topics:

Design-time Projects

A typical project is composed of several steps and milestones.

Some of these are:

- Define the business needs
- Identify and declare the sources and targets in the Topology
- Design and Reverse-engineer source and target data structures in the form of data models
- Implement data quality rules on these data models and perform static checks on these data models to validate the data quality rules
- Develop mappings using datastores from these data models as sources and target
- Develop additional components for tasks that cannot be achieved using mappings, such as Receiving and sending e-mails, handling files (copy, compress, rename and such), executing web services
- Integrate mappings and additional components for building Package workflows
- Version your work and release it in the form of scenarios
- Schedule and operate scenarios.

Oracle Data Integrator will help you cover most of these steps, from source data investigation to metadata lineage, and through loading and data quality audit. With its repository, Oracle Data Integrator will centralize the specification and development efforts and provide a unique architecture on which the project can rely to succeed.

"*Overview of an Integration Project*" in *Developing Integration Projects with Oracle Data Integrator* introduces you to the basic steps of creating an integration project with

Oracle Data Integrator. "*Creating an Integration Project*" in *Developing Integration Projects with Oracle Data Integrator* gives you more detailed information on the several steps of creating an integration project in ODI.

Batch-Oriented Integration

ODI is a comprehensive data integration platform with a built-in connectivity to all major databases, Big Data clusters, data warehouse and analytic applications providing high-volume and high-performance batch integration.

The main goal of a data warehouse is to consolidate and deliver accurate indicators to business users to help them make decisions regarding their everyday business. A typical project is composed of several steps and milestones. Some of these are:

- Defining business needs (Key Indicators)
- Identifying source data that concerns key indicators; specifying business rules to transform source information into key indicators
- Modeling the data structure of the target warehouse to store the key indicators
- Populating the indicators by implementing business rules
- Measuring the overall accuracy of the data by setting up data quality rules
- Developing reports on key indicators
- Making key indicators and metadata available to business users through adhoc query tools or predefined reports
- Measuring business users' satisfaction and adding/modifying key indicators

With its repository, ODI will centralize the specification and development efforts and provide a unique architecture on which the project can rely to succeed.

Scheduling and Operating Scenarios

Scheduling and operating scenarios is usually done in the Test and Production environments in separate Work Repositories. Any scenario can be scheduled by an ODI Agent or by any external scheduler, as scenarios can be invoked by an operating system command.

When scenarios are running in production, agents generate execution logs in an ODI Work Repository. These logs can be monitored either through the Operator Navigator or through any web browser when Oracle Data Integrator Console is setup. Failing jobs can be restarted and ad-hoc tasks submitted for execution.

E-LT

ODI uses a unique E-LT architecture that leverages the power of existing RDBMS and Big Data engines by generating native SQL and bulk loader control scripts to execute all transformations.

Event-Oriented Integration

Capturing events from a Message Oriented Middleware or an Enterprise Service Bus has become a common task in integrating applications in a real-time environment. Applications and business processes generate messages for several subscribers, or they consume messages from the messaging infrastructure.

Oracle Data Integrator includes technology to support message-based integration and that complies with the Java Message Services (JMS) standard. For example, a

transformation job within Oracle Data Integrator can subscribe and source messages from any message queue or topic. Messages are captured and transformed in real time and then written to the target systems.

Other use cases of this type of integration might require capturing changes at the database level. Oracle Data Integrator Changed Data Capture (CDC) capability identifies and captures inserted, updated, or deleted data from the source and makes it available for integration processes.

ODI provides two methods for tracking changes from source datastores to the CDC framework: triggers and RDBMS log mining. The first method can be deployed on most RDBMS that implement database triggers. This method is optimized to minimize overhead on the source systems. For example, changed data captured by the trigger is not duplicated, minimizing the number of input/output operations, which slow down source systems. The second method involves mining the RDBMS logs—the internal change history of the database engine. Log-based CDC is supported using Oracle GoldenGate.

The CDC framework used to manage changes, based on Knowledge Modules, is generic and open, so the change-tracking method can be customized. Any third-party change provider can be used to load the framework with changes.

Changes frequently involve several data sources at the same time. For example, when an order is created, updated, or deleted, both the orders table and the order lines table are involved. When processing a new order line, it is important that the new order, to which the line is related, is taken into account too. ODI provides a mode of change tracking called Consistent Set CDC. This mode allows for processing sets of changes for which data consistency is guaranteed.

For example, incoming orders can be detected at the database level using CDC. These new orders are enriched and transformed by ODI before being posted to the appropriate message queue or topic. Other applications such as Oracle BPEL or Oracle Business Activity Monitoring can subscribe to these messages, and the incoming events will trigger the appropriate business processes.

For more information on how to use the CDC framework in ODI, see "*Using Journalizing*" in *Developing Integration Projects with Oracle Data Integrator*.

Service-Oriented Architecture

Oracle Data Integrator can be integrated seamlessly in a Service-Oriented Architecture (SOA) in several ways:

Data Services are specialized Web services that provide access to data stored in database tables. Coupled with the Changed Data Capture capability, data services can also provide access to the changed records for a given subscriber. Data services are automatically generated by Oracle Data Integrator and deployed as Web services to a Web container, usually a Java application server. For more information on how to set up, generate and deploy data services, see "*Creating and Using Data Services*" in *Administering Oracle Data Integrator*.

Oracle Data Integrator can also expose its transformation processes as Web services to enable applications to use them as integration services. For example, a LOAD_SALES batch process used to update the CRM application can be triggered as a Web service from any service-compliant application, such as Oracle BPEL, Oracle Enterprise Service Bus, or Oracle Business Activity Monitoring. Transformations developed using ODI can therefore participate in the broader Service Oriented Architecture initiative.

Third-party Web services can be invoked as part of an ODI workflow and used as part of the data integration processes. Requests are generated on the fly and responses processed through regular transformations. Suppose, for example, that your company subscribed to a third-party service that exposes daily currency exchange rates as a Web service. If you want this data to update your multiple currency data warehouse, ODI automates this task with a minimum of effort. You would simply invoke the Web service from your data warehouse workflow and perform any appropriate transformation to the incoming data to make it fit a specific format. For more information on how to use web services in ODI, see "*Using Web Services*" in *Developing Integration Projects with Oracle Data Integrator*.

Data Quality with ODI

With an approach based on declarative rules, Oracle Data Integrator is the most appropriate tool to help you build a data quality framework to track data inconsistencies.

Oracle Data Integrator uses declarative data integrity rules defined in its centralized metadata repository. These rules are applied to application data to guarantee the integrity and consistency of enterprise information. The Data Integrity benefits add to the overall Data Quality initiative and facilitate integration with existing and future business processes addressing this particular need.

Oracle Data Integrator automatically retrieves existing rules defined at the data level (such as database constraints) by a reverse-engineering process. ODI also allows developers to define additional, user-defined declarative rules that may be inferred from data discovery and profiling within ODI, and immediately checked.

Oracle Data Integrator provides a built-in framework to check the quality of your data in two ways:

- Check data in your data servers, to validate that this data does not violate any of the rules declared on the datastores in Oracle Data Integrator. This data quality check is called a static check and is performed on data models and datastores. This type of check allows you to profile the quality of the data against rules that are not enforced by their storage technology.
- Check data while it is moved and transformed by a mapping, in a flow check that checks the data flow against the rules defined on the target datastore. With such a check, correct data can be integrated into the target datastore while incorrect data is automatically moved into error tables.

Both static and flow checks are using the constraints that are defined in the datastores and data models, and both use the Check Knowledge Modules (CKMs). For more information see "*Flow and Static Control*" in *Developing Integration Projects with Oracle Data Integrator*.

> **Tip:** Oracle Enterprise Data Quality (EDQ) can augment ODI's Data Quality capabilities. For more information about EDQ, refer to the Oracle Enterprise Data Quality documentation.

4

Managing Environments

This chapter provides information about managing the integration project environments.

This chapter includes the following topics:

Managing Environments

Integration projects exist in different environments during their lifecycle (development, test, production) and may even run in different environments in production (multiple site deployment). Oracle Data Integrator makes easier the definition and maintenance of these environments, as well as the lifecycle of the project across these environments using the Topology.

The Topology describes the physical and logical architecture of your Information System. It gives you a very flexible way of managing different servers, environments and agents. All the information of the Topology is stored in the master repository and is therefore centralized for an optimized administration. All the objects manipulated within Work Repositories refer to the Topology. That's why it is the most important starting point when defining and planning your architecture.

The Topology is composed of data servers, physical and logical schemas, and contexts.

Data servers describe connections to your actual physical application servers and databases. They can represent for example:

- Apache Hive
- An Oracle Instance
- An IBM DB2 Database
- A Microsoft SQL Server Instance
- A File System
- An XML File
- and so forth.

At runtime, Oracle Data Integrator uses the connection information you have described to connect to the servers.

Physical schemas indicate the physical location of the datastores (tables, files, topics, queues) inside a data server. All the physical schemas that need to be accessed have to be registered under their corresponding data server, physical schemas are used to prefix object names and access them with their qualified names. When creating a

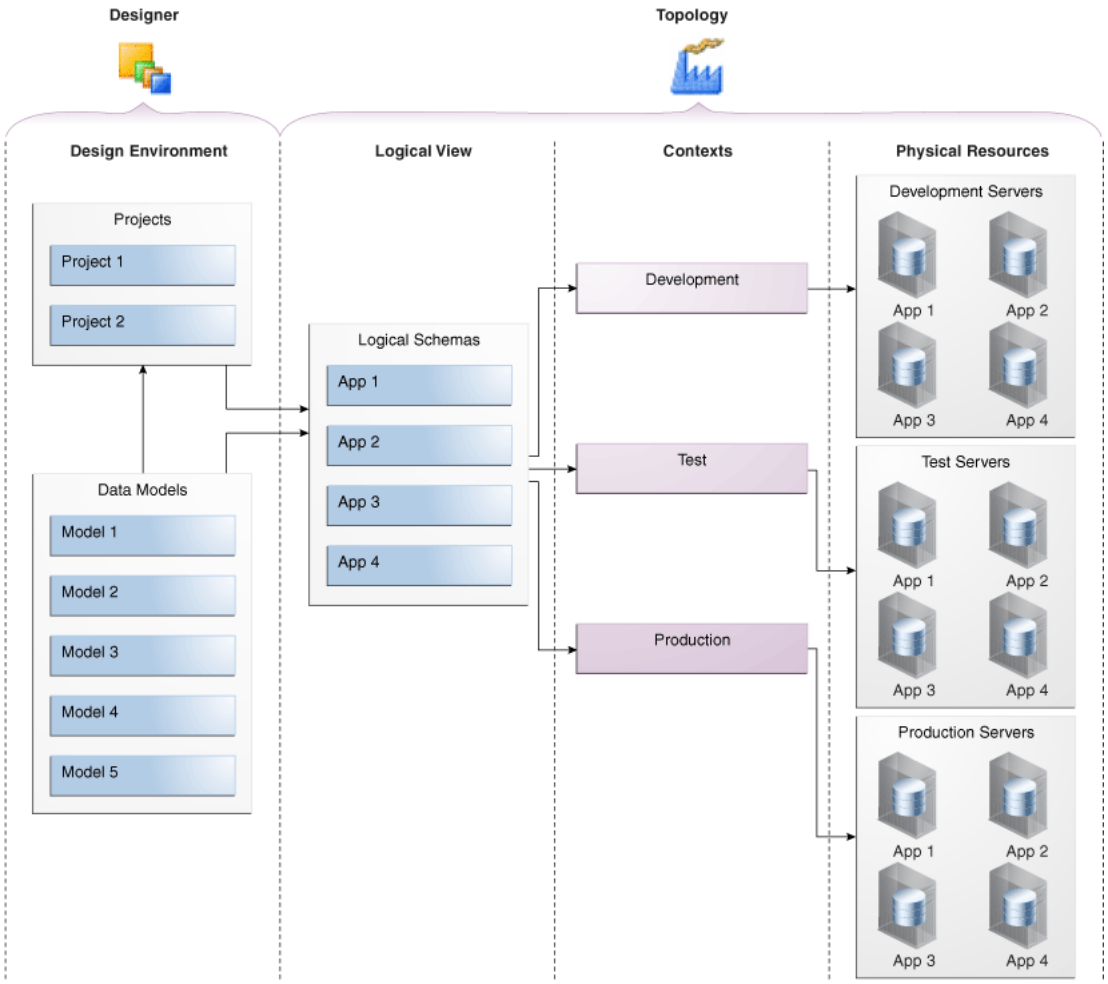
physical schema, you need to specify a temporary, or work schema that will store temporary or permanent objects needed at runtime.

A logical schema is an alias that allows a unique name to be given to all the physical schemas containing the same datastore structures. The aim of the logical schema is to ensure the portability of procedures and models on different design-time and run-time environments.

A Context represents one of these environments. Contexts are used to group physical resources belonging to the same environment.

Typical projects will have separate environments for Development, Test and Production. Some projects will even have several duplicated Test or Production environments. For example, you may have several production contexts for subsidiaries running their own production systems (Production New York, Production Boston, and so forth). There is obviously a difference between the logical view of the information system and its physical implementation as described in Figure 4–1.

Figure 4–1 Logical and Physical View of the Infrastructure



The logical view describes logical schemas that represent the physical schemas of the existing applications independently of their physical implementation. These logical schemas are then linked to the physical resources through contexts.

Designers always refer to the logical view defined in the Topology. All development done therefore becomes independent of the physical location of the resources they address. At runtime, the logical information is mapped to the physical resources, given the appropriate contexts. The same scenario can be executed on different physical servers and applications simply by specifying different contexts. This brings a very flexible architecture where developers don't have to worry about the underlying physical implementation of the servers they rely on.

5

Life Cycle Management in Oracle Data Integrator

Oracle Data Integrator integrates with Version Control Systems (VCS) to enable you to version control ODI objects.

This chapter includes the following topics:

Oracle Data Integrator integration with Version Control Systems

You can integrate ODI with an external Version Control Systems (VCS) to enable version control of ODI objects. You can store versioned copies of the ODI objects into an external VCS repository. As VCS relies on file-based storage, ODI objects are stored as XML files in the VCS repository.

Note: Currently ODI supports only Apache™ Subversion®.

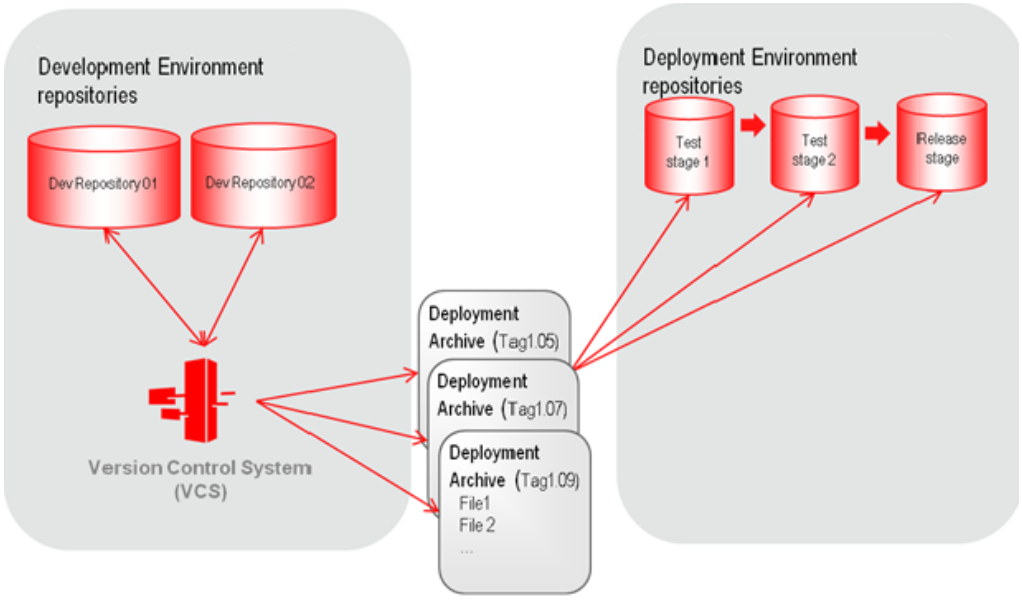
You can add un-versioned ODI objects to VCS, retrieve older versions of ODI objects from VCS, and view the differences between two versions of ODI objects from within ODI.

Fore more information, see *Integrating ODI with Version Control Systems* in *Developing Integration Projects with Oracle Data Integrator*.

Support for Separation of Deployment and Development Environments

ODI supports separation of the development environment and deployment environment, such as testing or production, where the ODI objects are not modified. ODI object updates in the deployment environments are now managed using Deployment Archives, which are the vehicle to release ODI objects from development to testing or production.

The following diagram depicts typical repositories in development and deployment environments.



The repositories in the development environment are connected to the version control system. These repositories are used to actively develop the ODI objects. Once the ODI objects are ready for testing or production, the deployment archives are created with them. These deployment archives are applied to a test repository for verification and upon successful testing are promoted to the production repositories.

The patches applied to the deployment environment using the deployment archives can be managed through the ODI UI, which allows you to audit the list of applied patches and also lets you to rollback any bad patches.

For detailed information, see *Release Management* in *Developing Integration Projects with Oracle Data Integrator*.