

An Oracle White Paper  
February 2009

# Oracle Data Integrator Enterprise Edition A Technical Overview

## Table of Contents

Introduction .....	2
E-LT Architecture .....	2
Traditional ETL.....	2
E-LT .....	3
Declarative DESIGN.....	5
Conventional ETL Design.....	5
Declarative Design .....	5
Knowledge modules .....	7
Knowledge Modules Type .....	7
Flexibility and Extensibility .....	8
Event-Oriented Integration.....	9
Changed Data Capture.....	9
Publish-and-Subscribe Model.....	10
Processing Consistent Sets of Changed Data .....	10
SOA Enabled.....	11
Data & Transformation Services.....	11
Web Services Access.....	12
Data Integrity.....	13
Declarative Rules for Data Integrity .....	13
Data Integrity Firewall in the Integration Process.....	14
Enforcing the Rules .....	15

Using Third-Party Name and Address Cleansing Tools .....	15
Architecture .....	15
User Interfaces .....	16
Agent .....	17
Repositories .....	17
Metadata Navigator / Lightweight Designer .....	18
Scenarios .....	18
Data Warehouse / Business Intelligence .....	18
Service-Oriented Integration .....	19
Master Data Management .....	19
Conclusion .....	20

## Introduction

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.

Many organizations address these diverse needs with a broad palette of tools and technologies, resulting in disjointed integration projects with no leverage or unity between them. These tools do not meet all performance, flexibility, and modularity requirements.

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

With powerful core differentiators - heterogeneous E-LT, Declarative Design and Knowledge Modules - ODI-EE meets the performance, flexibility, productivity, modularity and hot-pluggability requirements of an integration platform.

## E-LT Architecture

### Traditional ETL

Traditional ETL tools operate by first **E**xtracting the data from various sources, **T**ransforming the data on a proprietary, middle-tier ETL engine, and then **L**oading the transformed data onto the target data warehouse or integration server. Hence the term “ETL” represents both the names and the order of the operations performed.

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. Moreover, if one wants to ensure referential integrity by comparing data flow references against values from the target data warehouse, the

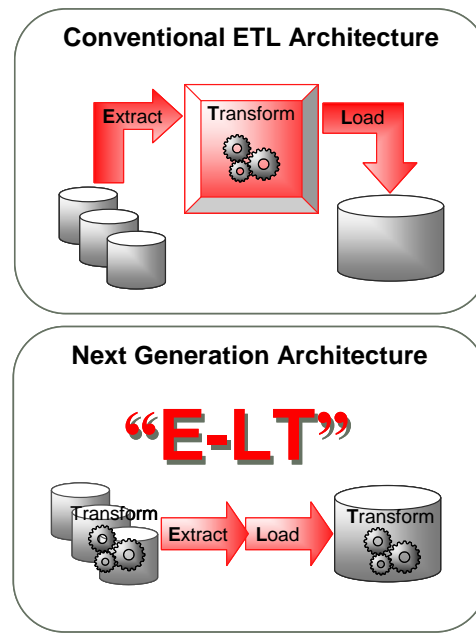
referenced data must be downloaded from the target to the engine, thus further increasing network traffic, download time, and leading to additional performance issues.

Let's consider, for example, how a traditional ETL job would look up values from the target database to enrich data coming from source systems. To perform such a job, a traditional ETL tool could be used in one of the following three ways:

- **Load look-up tables into memory:** The entire look-up table is retrieved from the target server and loaded into the engine's memory. Matching (or joining) this look-up data with source records is done in memory before the resulting transformed data is written back to the target server. If the look-up table is large, the operation will require a large amount of memory and a long time to download its data and re-index it in the engine.
- **Perform row-by-row look-ups "on the fly":** For every row, the ETL engine sends a query to the look-up table located on the target server. The query returns a single row that is matched (or joined) to the current row of the flow. If the look-up table contains, for example, 500,000 rows, the ETL engine will send 500,000 queries. This will dramatically slow down the data integration process and add significant overhead to the target system.
- **Use manual coding within the ETL job:** Use the ETL engine only for loading source data to the target RDBMS and manually write SQL code to join this data to the target look-up table. This raises the question: why buy an ETL tool that requires manual coding on the target server, knowing that you lose all the benefits of metadata management and development productivity by doing so? Unfortunately, this is what many users end up doing once they notice 10 x degradation in the overall performance of the integration process (when compared to the same operations executed by manual code).

## E-LT

The E-LT architecture incorporates the best aspects of both manual coding and ETL approaches in the same solution. The E-LT approach changes where and how data transformation takes place, and leverages the existing developer skills, RDBMS engines and server hardware to the greatest extent possible. In essence, E-LT relocates the data transformation step on the target RDBMS, changing the order of operations to: **E**xtract the data from the source tables, **L**oad the tables into the destination server, and then **T**ransform the data on the target RDBMS using native SQL operators.



### ETL vs. E-LT Approach

The E-LT architecture leverages the power of the RDBMS engine and throughput is only limited by the characteristics of the existing servers. Since no extra server, technology or skill requirement comes into play, the E-LT architecture provides optimal performance and scalability and eases the management of the integration infrastructure.

Let us see what would be the impact on the same lookup as described earlier;

- There is no need to excessively move data out of the source and target servers to the ETL server. Relevant source data would be moved to the target, and the lookup would directly occur in the same target server. This method reduces network traffic to the necessary part only.
- The processing will not be performed row-by-row, but with a single query using the join capabilities of the target server. Using the processing capabilities of the RDBMS engines provides the best performances for such an operation.
- The performances are therefore at least equal to those of manual coding, except that all the benefits of a data integration platform in terms of metadata management and productivity are still there.

With no separate ETL engine and no dedicated ETL server hardware required, the initial hardware and software capital costs are significantly lower, as are the ongoing software and hardware maintenance expenses. As the E-LT architecture uses any existing RDBMS to execute the ETL jobs, the overall cost is dramatically reduced. The same servers hosting the data are used to integrate them.

## Declarative DESIGN

### Conventional ETL 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

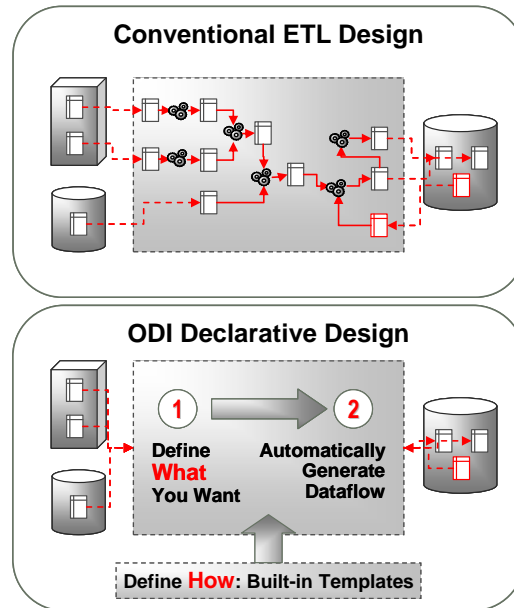
With Declarative Design, you just need to design *what* the process does, without describing *how* it will be done.

In our example, *what* the process does is:

- Relate the customer age from the sales application to the age groups from the statistical file.

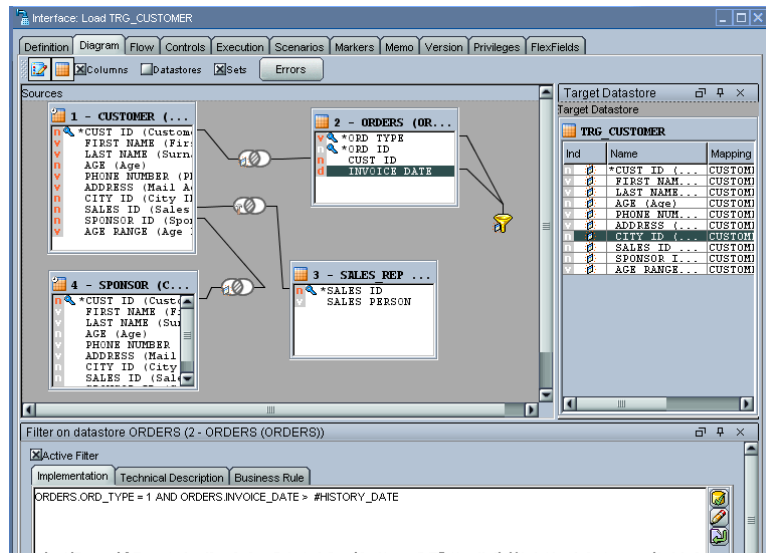
- Aggregate customer sales by age groups to load sales statistics.

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



### Conventional ETL Design vs. Declarative Design

Declarative Design in ODI-EE uses the well known relational paradigm to declare in the form of an “interface” the declarative rules for a data integration task, which includes designation of sources, targets, and transformations. In our example, the sources are the sales table and the age spreadsheet, and the target is the customer statistics system. The transformations are a join between the sales table and the age spreadsheet and an aggregation of the sales per age group.



### Designing a dataflow with Data Integrator

With declarative design, you focus on what really matters, that is the rules for your integration, instead of focusing on the underlying technical aspect of the integration process. The technical aspects are described in Knowledge Modules.

## Knowledge modules

### Knowledge Modules Type

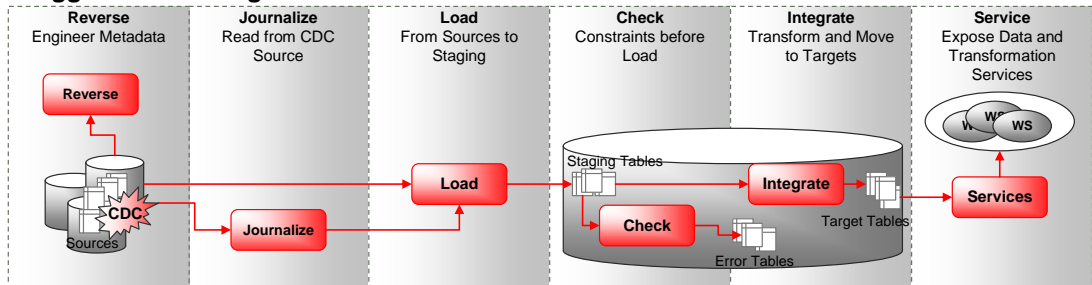
ODI-EE's Knowledge Modules implement *how* the integration processes occur.

Each Knowledge Module type refers to a specific integration task:

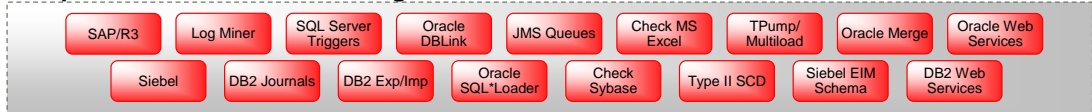
- Reverse-engineering metadata from the heterogeneous systems for ODI-EE
- Handling Changed Data Capture (CDC) on a given system.
- Loading data from one system to another, using system-optimized methods
- Integrating data in a target system, using specific strategies (insert/update, slowly changing dimensions)
- Controlling Data Integrity on the data flow
- Exposing data in the form of services.

These Knowledge Modules cover a wide range of technologies and techniques.

## Pluggable Knowledge Modules Architecture



## Sample out-of-the-box Knowledge Modules



## Knowledge Modules at Design and Runtime

A Knowledge Module is a code template for a given integration task. This code is independent of the interface, which defines the sources, targets and transformations that will be processed. At design-time, a developer creates metadata describing integration processes. This metadata is merged with the Knowledge Module to generate code ready for runtime. At runtime, ODI-EE sends this code for execution to the source and target systems it leverages for running the process.

## Flexibility and Extensibility

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 corresponds to Knowledge Modules tuned to exploit native capabilities of given platforms.

Knowledge modules are also fully extensible. Their code is opened 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.

ODI-EE comes out of the box with more than 100 Knowledge Modules for the major database engines and application packages of the market.

## Event-Oriented Integration

### Message-Oriented Integration

Capturing events from a Message Oriented Middleware or an Enterprise Service Bus such as Oracle ESB 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.

ODI-EE includes technology to support message-based integration and that complies with the Java Message Services (JMS) standard. For example, a transformation job within ODI-EE 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. The JMS model supports transactions to ensure data integrity and message delivery from the source middleware to the target systems.

Other use cases of this type of integration might require capturing changes at the database level and publishing them to the messaging infrastructure. The Changed Data Capture (CDC) capability of ODI-EE, described below, can be coupled with JMS-based integration, resulting in the highest flexibility for designing the required integration flows and mixing heterogeneous technologies.

For example, incoming orders can be detected at the database level using CDC. These new orders are enriched and transformed by ODI-EE 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.

### Changed Data Capture

The conventional data integration approach involves extracting all data from the source system, and then integrating the entire set—possibly incrementally—in the target system. This approach may reveal itself inefficient when the integration process reflects the need for event-oriented integration.

A typical example would be the need to propagate a new or updated contact file from a CRM application to another application. In this case, the amount of data involved—thousands of contacts—makes data integration impossible in the given timeframes.

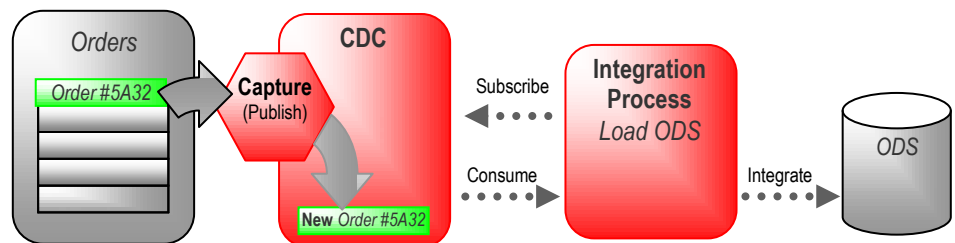
In-house designed solutions such as filtering records according to a timestamp column or “changed” flag are possible, but may require modifications in the

applications and are usually not sufficient for ensuring that all changes are properly taken into account.

ODI-EE Changed Data Capture (CDC) capability identifies and captures inserted, updated, or deleted data from the source and makes it available for integration processes. Changed Data Capture can be an alternative to Message-Oriented integration; it can also be used in conjunction with target-specific needs, minimizing development efforts.

### Publish-and-Subscribe Model

Changed Data Capture uses a publish-and-subscribe model. An identified subscriber—usually an integration process—subscribes to changes that happen in a datastore. Changes in the datastore are captured by the CDC framework and published for the subscriber, which can at any time process the tracked changes, and consume these events.



### Framework for tracking changes

ODI-EE 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. This has little impact on the system’s transactional performance and is supported for Oracle (through the Log Miner feature) and IBM DB2/400.

The CDC framework used to manage changes 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.

### Processing Consistent Sets of Changed Data

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-EE provides a mode of change tracking called Consistent Set CDC. This mode allows for processing sets of changes for which data consistency is guaranteed.

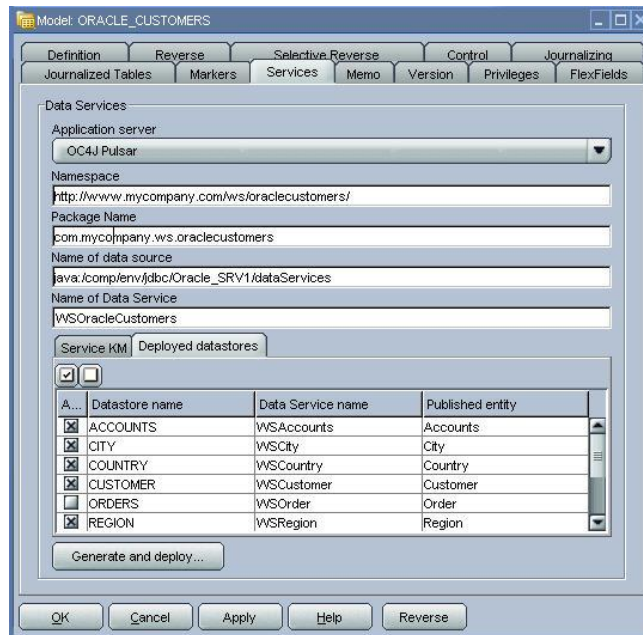
## SOA Enabled

ODI-EE plugs into the Oracle SOA Suite through three key service access capabilities: Data Services, Transformation Services and Web Service Access.

## Data & Transformation Services

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 ODI-EE and deployed as Web services to a Web container, usually a Java application server.

ODI-EE 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-EE can therefore participate in the broader Service Oriented Architecture initiative.

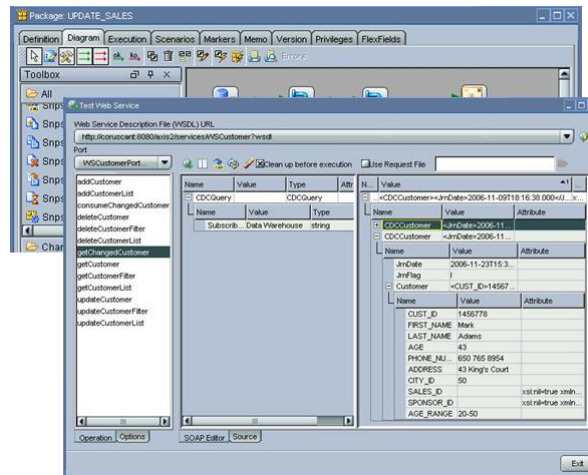


**Data Services Ready for Generation and Deployment**

## Web Services Access

Third-party Web services can be invoked as part of an ODI-EE 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-EE 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.



Using Web Services in work flows

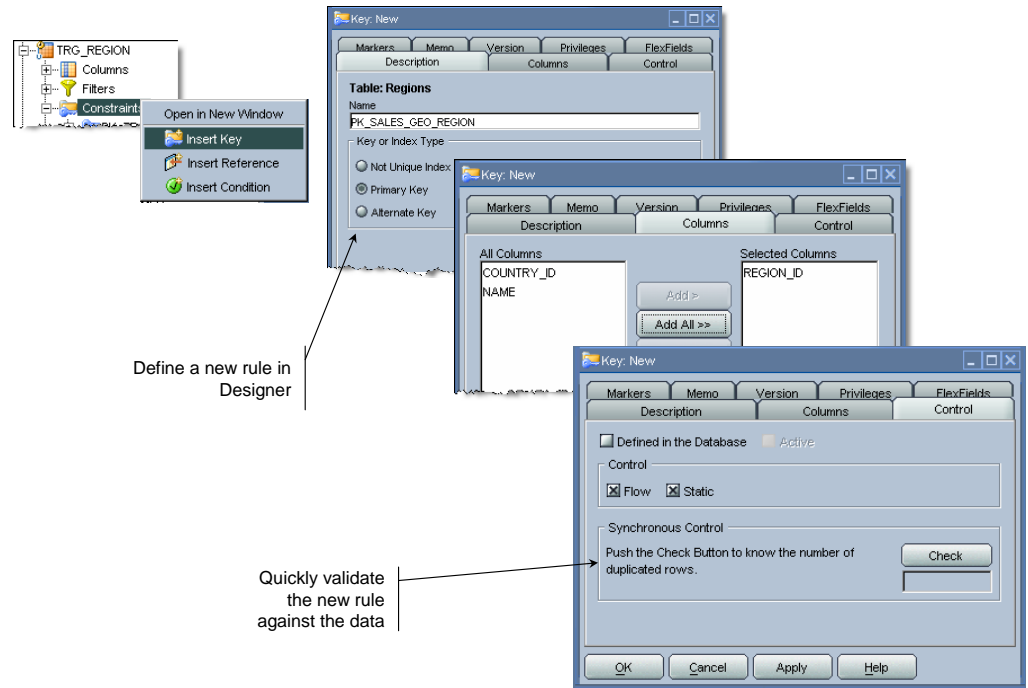
## Data Integrity

Data Integrity is the large subset of Data Quality which includes the processes that can be automated to ensure the quality of the enterprise information assets.

### Declarative Rules for Data Integrity

ODI-EE 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.

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

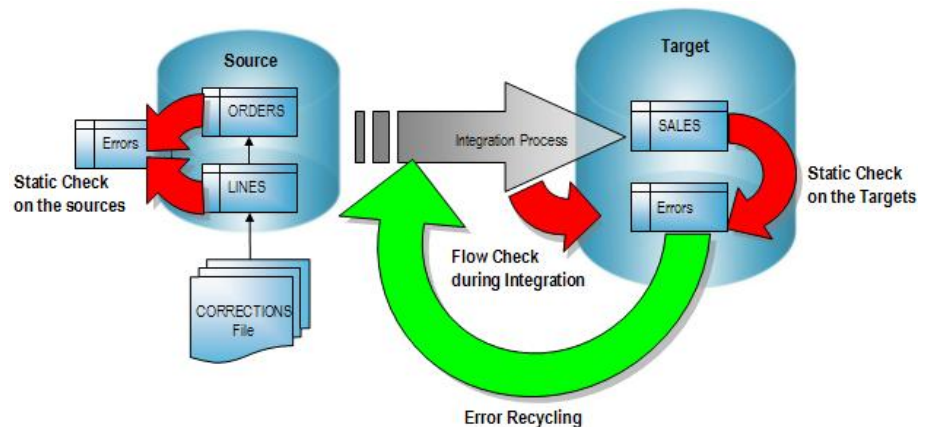


**Define a new integrity rule in Designer and immediately check it against the data.**

Declarative rules for data integrity include uniqueness rules, validation rules that enforce consistency at record level, and simple or complex reference rules possibly involving heterogeneous technologies.

### Data Integrity Firewall in the Integration Process

The components involved in an integration process are the source applications, the processes that transform and integrate data, and the target applications. With ODI-EE, data integrity is managed in all these subsystems, creating a real firewall that can be activated at any point of the integration process.

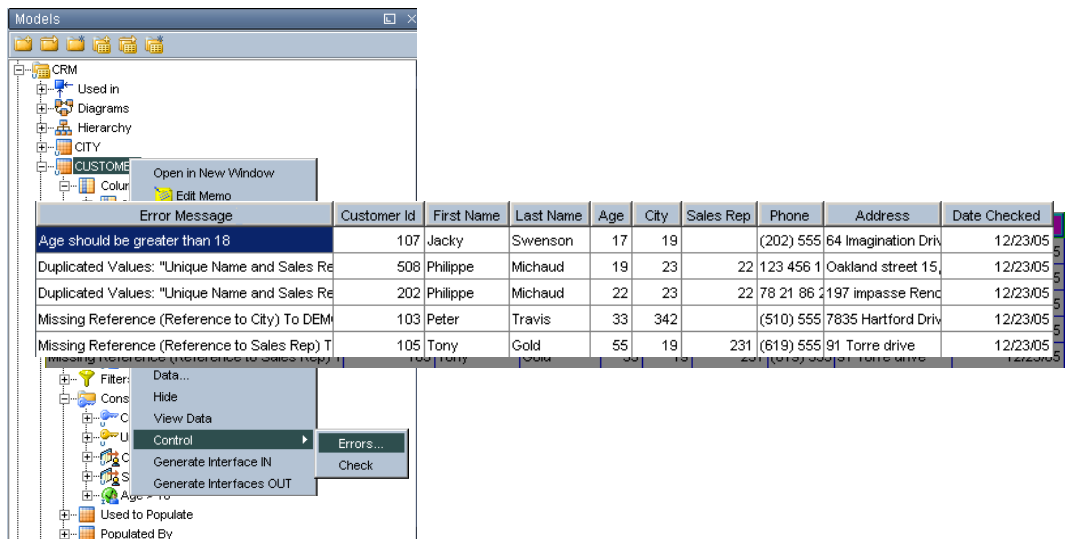


### Data Integrity in the Integration Process

## Enforcing the Rules

ODI-EE helps developers to automatically generate data integrity audits of their applications based on integrity rules that have been defined in the repository. Audits provide statistics on the quality of application data. They also isolate data that is detected as erroneous by applying the declarative rules. When erroneous records have been identified and isolated in error tables, they can be accessed from ODI-EE Designer or from any other front-end application.

This extensive information on data integrity makes possible detailed analysis, and takes erroneous data into account according to IT strategies and best practices for improving the overall data quality.



Error Message	Customer Id	First Name	Last Name	Age	City	Sales Rep	Phone	Address	Date Checked
Age should be greater than 18	107	Jacky	Swenson	17	19		(202) 555	64 Imagination Driv	12/23/05
Duplicated Values: "Unique Name and Sales Re	508	Philippe	Michaud	19	23	22	123 456 1	Oakland street 15,	12/23/05
Duplicated Values: "Unique Name and Sales Re	202	Philippe	Michaud	22	23	22	78 21 86 2	197 impasse Renc	12/23/05
Missing Reference (Reference to City) To DEM	103	Peter	Travis	33	342		(510) 555	7835 Hartford Driv	12/23/05
Missing Reference (Reference to Sales Rep) T	105	Tony	Gold	55	19		231 (619) 555	91 Torre drive	12/23/05

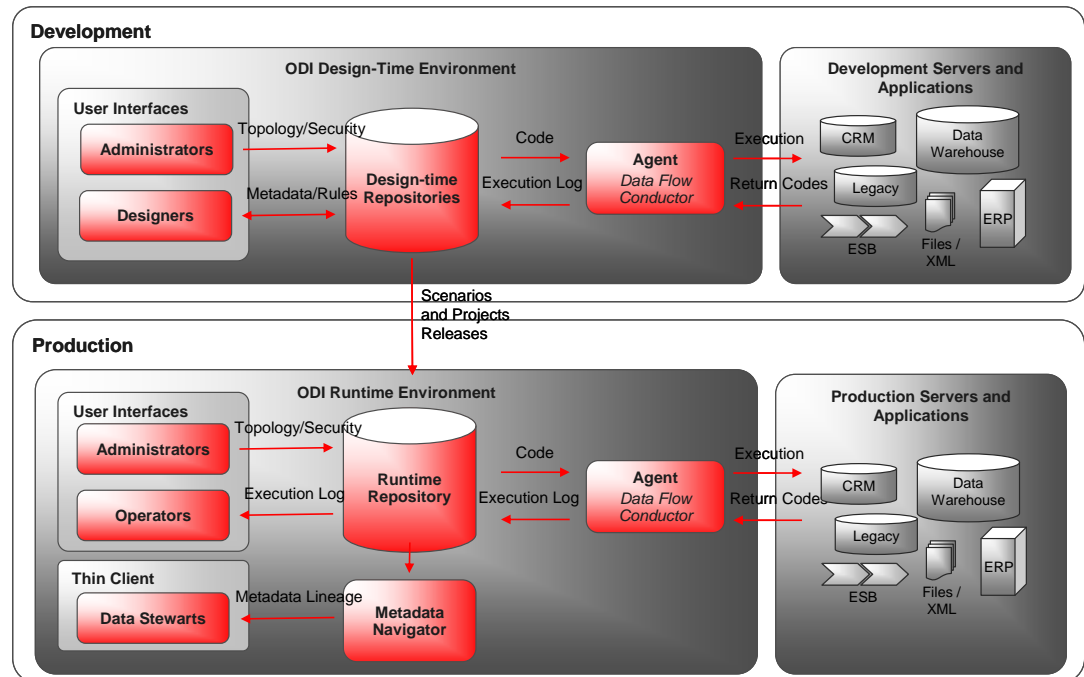
## Reviewing erroneous data in Designer

### Using Third-Party Name and Address Cleansing Tools

The ODI-EE open architecture can be seamlessly integrated with third-party data quality products, such as Trillium TS Quality to perform complex data quality and cleansing operations.

## Architecture

ODI-EE is organized around modular repositories. All run-time and design-time components are Java components that store their information in metadata repositories. The components of this architecture can be installed and run on any platform.



ODI-EE Detailed Architecture

## User Interfaces

The four ODI-EE graphical modules are:

- Designer:** In this interface, users can define declarative rules for data transformation and data integrity. Database and application metadata can be imported or defined. Designer uses metadata and rules to generate scenarios for production. All project development is performed through this interface, and it is the main user interface for developers and metadata administrators at design time.
- Operator:** In this interface, users can manage and monitor ODI-EE jobs in production. It is designed for production operators and shows the execution logs with error counts, the number of rows processed, execution statistics, the actual code that is executed, and so on. At design time, developers can also use Operator for debugging purposes. It is the main user interface at runtime.
- Topology Manager:** In this interface, users can define the physical and logical architecture of the infrastructure. Servers, schemas, and agents are registered in the ODI-EE master repository through this interface, which is primarily used by the administrators of the infrastructure or project.
- Security Manager:** In this interface, administrators can manage user accounts and privileges. It can be used to give profiles and users access rights

to ODI-EE objects and features. This interface is primarily used by security administrators.

## Agent

At runtime, the run-time component - the Agent - orchestrates the execution of scenarios. Execution may be launched from one of the user interfaces and triggered by the built-in scheduler or by a third-party scheduler.

Thanks to ODI-EE's E-LT architecture, the agent rarely performs any transformation itself. Usually, it simply retrieves code from the execution repository and requests that database servers, operating systems, or scripting engines execute that code. When the execution is completed, the agent updates the execution logs in the repository and reports error messages and execution statistics. The execution logs can be viewed from the Operator user interface or a web interface: Metadata Navigator.

It is important to understand that although it can act as a transformation engine, the agent is rarely used this way. Agents are installed at tactical locations in the information system to orchestrate the integration processes and leverage existing systems. Agents are multithreaded, load-balanced, lightweight components in this distributed integration architecture.

## Repositories

The ODI-EE repository is composed of a master repository and several work repositories. These repositories are pluggable in any relational database management systems (RDBMS).

All objects configured, developed, or used with ODI-EE modules are stored in one of these two types of repository.

There is usually only one master repository, which contains security information (user data and privileges), topology information (definitions of technologies and servers), and versions of objects.

The work repository is where projects are stored. Several work repositories may coexist in a single ODI-EE installation. This is useful for maintaining separate environments or to reflect a particular versioning lifecycle, for example, development, user acceptance tests, and production environments.

A work repository stores information related to:

- Models, including datastores, columns, data integrity rules, cross references, and data lineage
- Projects, including interfaces, packages, procedures, folders, knowledge modules, and variables
- Runtime information, including scenarios, scheduling information, and logs

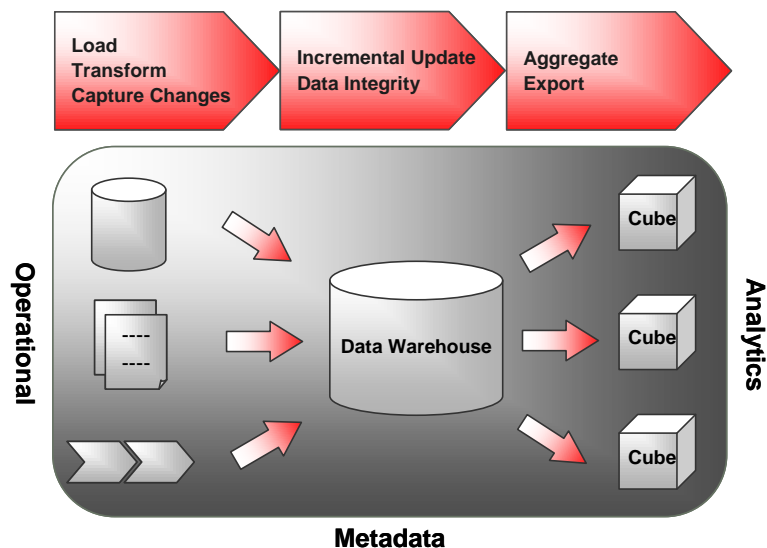
## Metadata Navigator / Lightweight Designer

Business users, developers, operators, and administrators can use their Web browser to access Metadata Navigator or Lightweight Designer. Through these Web interfaces, they can see flow maps, trace the source of all data and even drill down to the field level to understand the transformations used to build the data. They can also launch and monitor scenarios and edit data mappings through these web interfaces.

## Scenarios

ODI-EE can be used for all integration tasks. The following examples demonstrate typical situations for which the platform features can be exploited.

## Data Warehouse / Business Intelligence

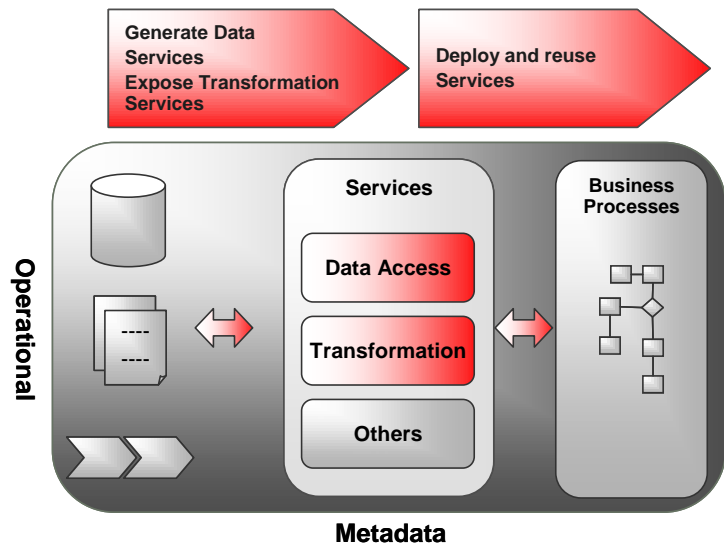


### Scenario: E-LT for Data Warehouse

In this common scenario, ODI-EE is used in batch mode to load the data warehouse from heterogeneous sources, using incremental updates or slowly changing dimensions (SCD type 2 or 3) strategies. Performance provided by the E-LT architecture enables narrow batch windows and leverages the power of the data warehouse server. Data integrity and consistency is ensured through the data integrity firewall. The Changed Data Capture feature updates in near real-time the data warehouse with changes occurring in operational data.

ODI-EE also aggregates and exports data from the data warehouse to load the OLAP cubes and analytics applications. Thanks to the comprehensive metadata supporting the integration processes, users can track data lineage from end to end.

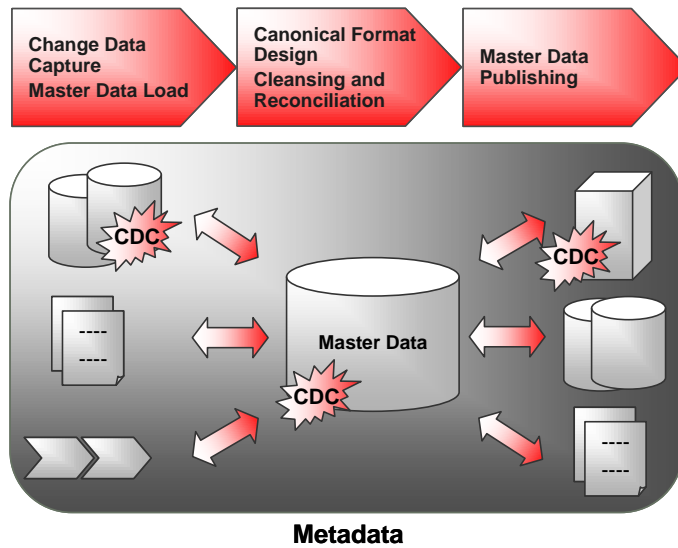
## Service-Oriented Integration



Scenario: SOA Initiative

In service-oriented development architectures, ODI-EE provides automatically generated services to access the data, as well as the transformations and integration processes it handles. Services available in the infrastructure can also be used in the transformation processes.

## Master Data Management



Scenario: Master Data Management

Users can define thanks to the intuitive Common Format Designer graphical feature the master data canonical format by assembling heterogeneous sources.

ODI-EE then generates the scripts to create the master data hubs as well as the data flows to reconcile and publish this master data from and to the operational systems.

ODI-EE can also be used in existing Master Data Management initiatives. It allows the synchronization of operational source systems with MDM applications such as Oracle Customer Data Hub and Oracle Product Information Management. Thanks to its Changed Data Capture framework, ODI-EE transforms and synchronizes data in real time from sources to MDM applications and from MDM applications to the subscribing operational applications. Data Integrity Firewall ensures master data quality and integrity.

Finally, the comprehensive and extensible metadata in the repository provides users with a solid foundation for their MDM initiative.

## Conclusion

Oracle Data Integrator Enterprise Edition fulfills the needs of Data Oriented, Event Driven and Service Oriented integration, and is fully adaptable to batch, real-time or synchronous latency requirements.

It provides key features to meet the pre-requisites for efficient data integration: Heterogeneous ETL for unmatched performance even in heterogeneous environments, Declarative Design for optimal productivity when designing transformations, Knowledge Modules providing the modularity and hot-pluggability to any system.



Oracle Data Integrator Enterprise Edition, A  
Technical Overview  
Feb 2009  
Author: F.X. Nicolas  
Contributing Authors: [OPTIONAL]

Oracle Corporation  
World Headquarters  
500 Oracle Parkway  
Redwood Shores, CA 94065  
U.S.A.

Worldwide Inquiries:  
Phone: +1.650.506.7000  
Fax: +1.650.506.7200  
oracle.com



Oracle is committed to developing practices and products that help protect the environment

Copyright © 2009, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.