Oracle9*i*AS Personalization

Recommendation Engine Batch API Programmer's Guide

Release 9.0.1

July 2001

Part No. A90091-01

ORACLE

Oracle9*i*AS Personalization Recommendation Engine Batch API Programmer's Guide, Release 9.0.1

Part No. A90091-01

Contents

2 RE Batch API Supporting Classes

3 Using the Recommendation Engine Batch Proxy

A REProxyBatch API Examples and Usage

B REProxyBatchAPI Example

Send Us Your Comments

Oracle9*i*AS Personalization Recommendation Engine Batch API Programmer's Guide, Release 9.0.1

Part No. A90091-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- DARWINDOC@us.oracle.com
- FAX: 781-684-7738. Attn: Oracle9*i*AS Personalization Documentation
- Postal service:
  Oracle Corporation
  Oracle9*i*AS Personalization Documentation
  200 Fifth Avenue
  Waltham, Massachusetts 02451
  U.S.A.

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This manual describes how a Java programmer can use Oracle9*i*AS Personalization (OP) Recommendation Engine Batch API (RE Batch API) to obtain recommendations.

Intended Audience

This manual is intended for Java programmers who create and maintain Web sites that use Oracle9*i*AS Personalization.

Structure

This manual contains three chapters and three appendixes:

Chapter 1	Introduces RE Batch API.
Chapter 2	Describes the RE Batch API supporting classes.
Chapter 3	Describes the methods used to manage sessions, manage data, and request recommendations
Appendix A	Contains examples of how to perform common tasks with the RE Batch API.
Appendix B	Contains complete sample program.

Where to Find More Information

The documentation set for Oracle9*i*AS Personalization at the current release consists of the following documents:

- README.htm, on the Oracle9*i*AS Personalization CD; this file contains platform-specific installation instructions.
- *Oracle9*i*AS Personalization Release Notes*, Release 9.0.1.
- *Oracle9*i*AS Personalization Administrator's Guide*, Release 9.0.1 (includes installation instructions that are the same across all platforms).
- *Getting Started with Oracle9*i*AS Personalization*, Release 9.0.1.
- *Oracle9*i*AS Personalization Recommendation Engine API Programmer's Guide*, Release 9.0.1. A programmer's manual for programming the recommendation engines in real time.
- *Oracle9*i*AS Personalization Recommendation Engine Batch API Programmer's Guide*, Release 9.0.1. A programmer's manual for obtaining bulk recommendations (this document).

Related Manuals

For more information about the database underlying OP, see:

- *Oracle9*i *Administrator's Guide*
- *Oracle9*i *Application Server Installation Guide* (the appropriate version for your operating system).

Requirements

OP documentation is distributed on the same CD that OP is distributed on. Documentation is provided in PDF and HTML formats.

After OP is installed, the OP documentation can be read by opening the following URL using either Netscape or Internet Explorer:

```
http://server/opDoc/op.901/index.htm
```

where *server* is name of the system where OP is installed.

You can read or print the documentation directly from the CD or from your browser.

To view the PDF files, you will need

- Adobe Acrobat Reader 3.0 or later, which you can download from `www.adobe.com`.

To view the HTML files, you will need

- Netscape 4.x or later, or
- Internet Explorer 4.x or later

Documentation Accessibility

Oracle's goal is to make our products, services, and supporting documentation accessible to the disabled community with good usability. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Conventions

In this manual, Windows refers to the Windows 95, Windows 98, and the Windows NT operating systems.

The SQL interface to Oracle9i is referred to as SQL. This interface is the Oracle 9i implementation of the SQL standard ANSI X3.135-1992, ISO 9075:1992, commonly referred to as the ANSI/ISO SQL standard or SQL92.

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The following conventions are also used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
....	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.
syntax text	Text in syntax font is used to describe the syntax of Java code.
italic text or *syntax*	Text or syntax in italics specify user-supplied names or data.

1

Introduction

The OP (Oracle9*i*AS Personalization) RE Batch API (Recommendation Engine Batch Application Programming Interface) enables a web application written in Java to request Oracle9*i*AS Personalization-style recommendations in bulk mode.

RE Batch API was designed to be extensible, to minimize the number of API functions, to be uniform, and to keep the number of arguments to a minimum.

Appendix A contains examples of how to perform common tasks using RE Batch API.

Appendix B contains a complete example of RE Batch API usage.

> **Note:** RE Batch API is installed on the system where Oracle9*i*AS is installed.

RE Batch API Prerequisites

Before you can use RE Batch API methods, OP must be installed and the appropriate tables must be created and populated. Your database tables must be converted to the OP schemas. It is important that the OP MTR is populated with customer profiles. You should also create tables or views containing the customer IDs for which you want recommendations.

If you are using one or more taxonomies, they must be properly specified.

At least one OP package must have been built and deployed. Use the OP administrative interface to do this. For an example of how to create and deply a package, see *Getting Started with Oracle9iAS Personalizaton*.

For detailed information about how to install OP, see the administration guide.

Definitions and Concepts

This section describes the collections of methods that make up the RE Batch API and concepts and terms used in the description of the API.

End Users (Customers)

End users (users of a Web site that uses OP for recommendations) are divided into two groups: customers and visitors. A *customer* is a registered user, who can be identified by a unique customer ID assigned by the web application. The RE Batch API makes recommendations for customers only.

Making Recommendations

Recommendations are based on historical data, which is stored in the database and retrieved when the customer profiles are loaded.

Using RE Batch API

Before you execute an RE Batch program, you must

- Set up the OP environment (create an RE, and create and deploy an OP package)
- Create the tables used by the RE Batch methods

Setting Up the OP Environment

Before you can execute RE Batch API methods, the following must be true:

- Properly formatted customer profile data must be available in the Mining Table Repository (MTR)
- A recommendation engine (RE) farm containing at least one recommendation engine must exist.
- A package must have been successfully built and then deployed in the recommendation engine farm.

The OP administration guide and the online help for the OP administrative GUI explain how to perform these steps.

Customer Profile Data

Customer profile data resides in the MTR.

Deploy a Package to an RE

You cannot get recommendations until there is an existing deployed package, which is created using the OP Administrative interface. You must build a package before you deploy it. You cannot build a package until there is some data available; data is converted from existing data collected by your web application and stored in an Oracle database.

When you design an OP application, you must decide if there should be more than one RE and, if there are several REs, how to use them. We recommend that the REs used for bulk recommendations not be used for any other purpose. For a discussion of the design considerations, see "Recommendation Engine Usage" in Appendix A.

Note: If you try to deploy a package an RE while a batch program is running, the deployment will fail.

Recommendations may want to take income level (salary) into consideration; for example, you may want to recommend items that the user can afford to buy. If the items that are recommended have prices in several currencies (for example, items are sold in Japan and India), see "Handling Multiple Currencies" in Appendix A.

Sample RE Batch API Usage

OP includes a sample Java program that illustrates the use of many of the RE Batch API methods; the program is in Appendix B. There are also some examples of how to perform typical tasks in Appendix A.

Creating an REBatchProxy Object

Before you can use any of the RE Batch API methods, you must create at least one REBatchProxy object; see Appendix A for details. The object establishes a JDBC connection to a specified database and schema. The connection exists until it is explicitly destroyed.

Creating Instances of Objects

To use the API, you must create instances of the objects used by the API method signatures. Use the RE Batch API supporting classes, described in Chapter 2, to create these instances. It is always necessary, for example to create filtering settings and tuning sessions. For examples, see Appendix A.

Converting Data

OP generates recommendations based on data describing past user behavior.

User data stored in an Oracle table must be transformed and stored in the Mining Table Repository (MTR) before it can be used to generate recommendations.

Managing Customer Profiles

OP stores customer profiles in the Mining Table Repository (MTR). The profiles to be used must be loaded into an RE before any recommendation requests are made. The following methods manage load and unload customer profiles from an RE:

- loadCustomerProfiles()
- purgeCustomerProfiles()

Before you load a set of customer profiles, you must create a table or a view containing a list of the customer IDs that identify the profiles that you wish to load, that is, a list of the customer IDs for which you want a recommendation.

Getting Recommendations

To get a recommendation, the application calls one of the following recommendation methods:

- crossSellForItem()
- rateItem()
- recommendTopItems()

These methods are used for getting recommendations for customers (registered users).

Ratings in OP

Ratings in OP are in "ascending order of goodness", that is, the higher the rating, the more the user prefers the item. Low rated items are items that the user does not prefer. OP algorithms use these assumptions, so it is important that ratings are in ascending order of goodness.

Creating Recommendations

OP uses rule tables stored in the RE to calculate the recommendations requested by the methods listed above. The specific rule table used depends upon the RE Batch API method used. In general, the antecedents of the rules are matched against the

historical data and the probabilities of the various consequents are computed. These items are then ordered by probability, and numberOfItems (an API argument) items are returned. The recommendations are written to a database table.

If there is enough memory in the RE database, the RE caches all rules associated with a particular package deployed from the MTR to the RE, not just the most recent rules.

Scoring: For scoring, all available historical data is used.

The OP Mining Table Repository (MTR) contains historical rating, transactional data, and navigational data stored in both detailed and aggregated formats. The MTR also contains demographic data. When scoring for customers, the RE retrieves the demographic data and the aggregated version of the other data source types.

Making Recommendations

RE Batch API methods that make recommendations write the recommendations to a database table. The schema used for the output depends on the method used. You can extract the recommendations in many ways, for example, with an appropriate SQL query, and then decide which recommendations to pass to the user.

Removing the REBatchProxy Object

Before you exit the application, you should destroy any proxy objects that you created.

2

RE Batch API Supporting Classes

This chapter describes the supporting classes for the REBatchProxy class. These classes are used to create instances of the objects used by the methods described in Chapter 3. You may be able to create one instance of many of these classes and use that one instance as an argument for several calls.

> **Note:** Except for Location, these supporting classes are the same as the ones that are used by the OP real-time (REAPI) recommendations.

Before you issue any of the recommendation methods described in Chapter 3, you must generate appropriate FilteringSettings, TuningSettings, and Location instances.

All methods described in this chapter are public.

The supporting classes are divided into two categories:

- EnumType interfaces
- Other supporting classes

Ratings in OP

Ratings in OP are in "ascending order of goodness", that is, the higher the rating, the more the user prefers the item. Low rated items are items that the user does not prefer. OP algorithms use these assumptions, so it is important that ratings are in ascending order of goodness.

Location of Classes

The following frequently used classes are in the oracle.dmt.re.base subdirectory:

- DataItem
- Enum
- FilteringSettings
- TuningSettings

For example, to use the Enum interfaces, you must include the following statement in your Java program:

```
import oracle.dmt.op.re.base.Enum;
```

EnumType Interfaces

Many of the RE Batch API methods reference attributes that can take on a finite number of values. The interface Enum is used to implement the base class for these "enumerations."

The Enum interface has a nested EnumType class with the following general methods:

```
int getId();

String toString();

String getName();

boolean isEqual(EnumType);
```

The following interfaces extend EnumType:

- CategoryMembership
- DataSource
- InterestDimension
- PersonalizationIndex
- ProfileDataBalance
- ProfileUsage
- Sorting

CategoryMembership Interface

CategoryMembershipType is implemented as:

- CategoryMembershipType (a class that extends EnumType)
- CategoryMembership (an interface)

The class CategoryMembership has the following methods:

```
CategoryMemberShipType getType(String name);

CategoryMemberShipType getType(int);
```

CategoryMembership specifies how categories in a list of categories should be applied for filtering. For example, Enum.CategoryMembership.EXCLUDE_ITEMS specifies that items from the category should be excluded from the category list. For details, see FilteringSettings later in this chapter.

CategoryMembership takes on the following values:

- Enum.CategoryMembership.EXCLUDE_ITEMS
- Enum.CategoryMembership.INCLUDE_ITEMS
- Enum.CategoryMembership.EXCLUDE_CATEGORIES
- Enum.CategoryMembership.INCLUDE_CATEGORIES
- Enum.CategoryMembership.LEVEL
- Enum.CategoryMembership.SUBTREE_ITEMS
- Enum.CategoryMembership.SUBTREE_CATEGORIES
- Enum.CategoryMembership.ALL_ITEMS
- Enum.CategoryMembership.ALL_CATEGORIES

The following statement assigns Enum.CategoryMembership.LEVEL to the variable myEnum:

```
CategoryMembershipType myEnum = Enum.CategoryMembership.LEVEL;
```

DataSource Interface

DataSource is implemented as:

- DataSourceType (a class that extends EnumType)
- DataSource (an interface)

The class DataSourceType has the following methods:

```
DataSourceType getType(String name);

DataSourceType getType(int);
```

DataSource specifies the type of data that is used when OP performs certain operations. For example, Enum.DataSource.DEMOGRAPHIC specifies that demographic data. The method DataItem, described later in this chapter, uses DataSource. Note that a given method may not support all values of DataSource. For details, see the description of the method in Chapter 3.

DataSource takes on the following values:

- Enum.DataSource.DEMOGRAPHIC
- Enum.DataSource.PURCHASING
- Enum.DataSource.RATING
- Enum.DataSource.NAVIGATION
- Enum.DataSource.ALL

The following statement assigns Enum.DataSource.ALL to the variable myEnum:

```
DataSourceType myEnum = Enum.DataSource.ALL;
```

InterestDimension Interface

InterestDimension is implemented as:

- InterestDimensionType (a class that extends EnumType)
- InterestDimension (an interface)

The class InterestDimensionType has the following methods:

```
InterestDimensionType getType(String name);

InterestDimensionType getType(int);
```

InterestDimension indicates the type of interest that the user of the web site has in a given item. NAVIGATION indicates that the user is interested in the items. PURCHASING indicates that the user would like to purchase the items. RATING indicates that the user likes the items. For more information, see the description of the TuningSettings method later in this chapter.

InterestDimension takes on the following values:

- Enum.InterestDimension.NAVIGATION

- Enum.InterestDimension.PURCHASING
- Enum.InterestDimension.RATING

The following statement assigns Enum.InterestDimension.PURCHASING to the variable myEnum:

```
InterestDimensionType myEnum = Enum.InterestDimension.PURCHASING;
```

PersonalizationIndex Interface

PersonalizationIndex is implemented as:

- PersonalizationIndexType (a class that extends EnumType)
- PersonalizationIndex (an interface)

The class PersonalizationIndexType has the following methods:

```
PersonalizationIndexType getType(String name);

PersonalizationIndexType getType(int);
```

PersonalizationIndex specifies how "unusual" the recommendations returned will be. For example, LOW specifies not unusual. For more information, see the description of the TuningSettings method later in this chapter.

PersonalizationIndex takes on the following values:

- Enum.PersonalizationIndex.LOW
- Enum.PersonalizationIndex.MEDIUM
- Enum.PersonalizationIndex.HIGH

The following statement assigns Enum.PersonalizationIndex.LOW to the variable myEnum:

```
PersonalizationIndexType myEnum = Enum.PersonalizationIndex.LOW;
```

ProfileDataBalance Interface

ProfileDataBalance is implemented as:

- ProfileDataBalanceType (a class that extends EnumType)
- ProfileDataBalance (an interface)

The class ProfileDataBalanceType has the following methods:

```
ProfileDataBalanceType getType(String name);
```

```
ProfileDataBalanceType getType(int);
```

ProfileDataBalance specifies whether to take data from the current session or from history or to balance data between data from the current session and history when making recommendations. For more information, see the description of the TuningSettings method later in this chapter.

ProfileDataBalance takes on the following values:

- Enum.ProfileDataBalance.HISTORY

> **Note:** The only value of profile data balance that makes sense for bulk recommendations is Enum.ProfileDataBalance.HISTORY. You must specify this value. (There is no current session data available.)

The following statement assigns Enum.ProfileDataBalance.HISTORY to the variable myEnum:

```
ProfileDataBalanceType myEnum = Enum.ProfileDataBalance.HISTORY;
```

ProfileUsage Interface

ProfileUsage is implemented as:

- ProfileUsageType (a class that extends EnumType)
- ProfileUsage (an interface)

The class ProfileUsageType has the following methods:

```
ProfileUsageType getType(String name);

ProfileUsageType getType(int);
```

ProfileUsage specifies whether the recommendation list can include or exclude items in a customer's profile. For more information, see the description of TuningSettings later in this chapter.

ProfileUsage takes on the following values:

- Enum.ProfileUsage.INCLUDE
- Enum.ProfileUsage.EXCLUDE

The following statement assigns Enum.ProfileUsage.INCLUDE to the variable myEnum:

```
ProfileUsageType myEnum = Enum.ProfileUsage.INCLUDE;
```

Sorting Interface

Sorting is implemented as:

- SortingType (a class that extends EnumType)
- Sorting (an interface)

The class SortingType has the following methods:

```
SortingType getType(String name);

SortingType getType(int);
```

Sorting indicates whether sorting is done (none implies no sorting), and, if sorting is done, how it is done (ascending or descending). For more information, see the discussion of the DataItem class later in this chapter.

Sorting takes on the following values:

- Enum.Sorting.NONE
- Enum.Sorting.DESCENDING
- Enum.Sorting.ASCENDING

The following statement assigns Enum.Sorting.NONE to the variable myEnum:

```
SortingType myEnum = Enum.Sorting.NONE;
```

Other Supporting Classes

The other supporting classes are

- DataItem
- FilteringSettings
- Location
- TuningSettings

DataItem

A subclass of class Item. This class encapsulates data about an item.

Attributes

No public attributes.

Methods

There are two kinds of methods provided with this class:

- A constructor that creates a DataItem instance
- Methods that return attribute values

The following constructor creates a data item. For a description of an attribute, see the description of the corresponding method that returns the attribute value.

```
//Constructor
DataItem(String type, long ID, DataSourceType dataSource, String value);
```

Create a DataItem instance for a given item. An item is uniquely identified by its type and ID. For a description of an attribute, see the description of the method that returns that attribute value.

The remaining methods return attribute values:

getDataSource();

Returns a value of type DataSourceType representing the type of data associated with the item; the values supported are

- Enum.DataSource.DEMOGRAPHIC
- Enum.DataSource.PURCHASING
- Enum.DataSource.RATING
- Enum.DataSource.NAVIGATION

getValue();

Returns an value of type String that represents the value associated with the item.

Usage Notes

dataSource cannot be Enum.DataSource.ALL.

FilteringSettings

Specifies the items to include or exclude when generating recommendations. Release 1 of OP supports category filtering only.

Attributes

No public attributes.

Methods

There are three kinds of methods provided with this class:

- A constructor for FilteringSettings
- Methods that set the attributes values
- Methods that return attribute values

```
//Constructor
FilteringSettings (int taxonomyID);
```

Creates an instance with CategoryFiltering set to Enum.Filtering.OFF, CategoryMembership set to Enum.CategoryMembership.ALL_ITEMS, and taxonomyID set to the provided integer.

The following methods set the attributes of a FilteringSettings instance. First you must use the default constructor to create an instance. For a description of an attribute, see the description of the method that returns that attribute value.

```
setItemFiltering(int taxonomyID);
```

Creates an instance with CategoryFiltering set to Enum.Filtering.ON, CategoryMembership set to Enum.CategoryMembership.ALL_ITEMS, categoryList set to null, and taxonomyID set to the provided integer. Use this method to recommend items from *all* leaves in the taxonomy with the specified ID.

```
setItemFiltering(int taxonomyID, long[] categoryList);
```

Creates an instance with CategoryFiltering set to Enum.Filtering.ON, CategoryMembership set to Enum.CategoryMembership.INCLUDE_ITEMS, taxonomyID set to the provided integer, and categoryList set to the provided array. Use this method to recommend items that belong to the categories in the categoryList argument and the taxonomy with the specified ID.

```
setItemExclusion(int taxonomyID, long[] categoryList);
```

Creates an instance with CategoryFiltering set to Enum.Filtering.ON, CategoryMembership set to Enum.CategoryMembership.EXCLUDE_ITEMS, taxonomyID set to the provided integer,

and categoryList set to the provided array. Use this method to recommend those items in the taxonomy with the specified ID that do *not* belong to the categories in the categoryList argument.

setItemSubTreeFiltering(int taxonomyID, long[] categoryList);

Creates an instance with CategoryFiltering set to Enum.Filtering.ON, CategoryMembership set to Enum.CategoryMembership.SUBTREE_ITEMS, taxonomyID set to the provided integer, and categoryList set to the provided array. Use this method to recommend those items in the taxonomy with the specified ID that belong to the subtrees of the categories in the categoryList argument.

setCategoryExclusion(int taxonomyID, long[] categoryList);

Creates an instance with CategoryFiltering set to Enum.Filtering.ON, CategoryMembership set to Enum.CategoryMembership.EXCLUDE_CATEGORIES, categoryList set to the provided array, and taxonomyID set to the provided integer. Use this method to recommend those categories in the taxonomy with the specified ID that are *not* in the category list.

setCategorySubTreeFiltering(int taxonomyID, long[] categoryList);

Creates an instance with CategoryFiltering set to Enum.Filtering.ON, CategoryMembership set to Enum.CategoryMembership.SUBTREE_CATEGORIES, taxonomyID set to the provided integer, and categoryList set to the provided array. Use this method to recommend categories in the taxonomy with the specified ID that belong to the subtrees of the categories in the categoryList argument.

setCategoryLevelFiltering(int taxonomyID, long[] categoryList);

Creates an instance with CategoryFiltering set to Enum.Filtering.ON, CategoryMembership set to Enum.CategoryMembership.LEVEL, taxonomyID set to the provided integer, and categoryList set to the provided array. Use this method to recommend categories in the taxonomy with the specified ID that belong to the same levels as the categories in the categoryList argument.

setCategoryFiltering(int taxonomyID);

Creates an instance with CategoryFiltering set to Enum.Filtering.ON, CategoryMembership set to Enum.CategoryMembership.ALL_CATEGORIES, and taxonomyID set to the provided integer. Use this method to recommend categories from all categories in the taxonomy with the specified ID.

setCategoryFiltering(int taxonomyID, long[] categoryList);

Creates an instance with CategoryFiltering set to Enum.Filtering.ON, CategoryMembership set to INCLUDE _CATEGORIES, taxonomyID set to the provided integer, and categoryList

set to the argument provided. Use this method to recommend categories in the taxonomy with the specified ID that belong to the subtrees of the categories in the categoryList argument.

The following methods return the FilteringSettings attributes:

getTaxonomyID();

Returns the integer (type int) that identifies the taxonomy to which the categories belong.

getCategoryFiltering();

Returns the value of Category Filtering (type FilteringType). The options are

- Enum.Filtering.ON — perform category filtering
- Enum.Filtering.OFF— do not perform category filtering

getCategoryList();

Returns the array of 64-bit integers (type long[]) that indicates the list of categories to be used as a filter for recommending categories or items.

getCategoryMembership();

Returns a value of type CategoryMembershipType that specifies how the categories in the category list should be applied for filtering. Options are

- Enum.CategoryMembership.EXCLUDE_ITEMS — exclude items that belong to categories in the category list
- Enum.CategoryMembership.EXCLUDE_CATEGORIES — exclude categories in the category list
- Enum.CategoryMembership.INCLUDE_ITEMS — include items that belong to categories in the category list
- Enum.CategoryMembership.INCLUDE_CATEGORIES — include categories in the category list
- Enum.CategoryMembership.LEVEL — recommend categories from the same level as the category in the list
- Enum.CategoryMembership.SUBTREE_ITEMS — recommend items that belong to levels in the category list below those of the categories in the category list
- Enum.CategoryMembership.SUBTREE_CATEGORIES — recommend categories that belong to the subtrees in the category list below the categories in the category list.

- Enum.CategoryMembership.ALL_ITEMS — include all items in the specified taxonomy
- Enum.CategoryMembership.ALL_CATEGORIES — include all categories in the specified taxonomy

Usage Notes

Not all filtering settings can be used will all methods. In particular, the following filtering setting cannot be used with the cross-sell methods (crossSellForItems):

- setCategoryLevelFiltering
- setCategorySubtreeFiltering
- setCategoryExclusion
- setCategoryFiltering(int)
- setCategoryFiltering(int, long[])

Item

This class is used to represent items that can be recommended and for which data can be collected. An item is uniquely represented by the combination of type and ID. Item IDs must be unique within a given type, but different types can have the same IDs.

Attributes

No public attributes.

Methods

There are two kinds of methods provided with this class:

- A constructor that creates an Item instance
- Methods that return attribute values

The following method creates an Item instance:

```
//Constructor
Item(String type, long ID);
```

Creates an instance with the specified type and ID. For descriptions of type and ID, see the descriptions of the methods that return them.

The following methods return attributes:

```
getType();
```

Returns a value of type String that identifies the group to which an item belongs. For example, a web site might have two types of items: products and banner ads. Individual products will have unique IDs, as will individual banner ads.

```
getID();
```

Returns an number of type long that is the unique identifier for an item within a given type.

Usage Note

Different items in a given type must have different IDs.

Location

Specifies the location of the input table or the table containing the results of an REBatchProxy method are stored. The schema for the table depends on the call made. See the descriptions of the individual methods in Chapter 3 for details.

Attributes

No public attributes.

Methods

There are three kinds of methods provided with this class:

- A constructor that creates a Location instance
- Methods that return attribute values
- Methods that set attribute values

//constructor
Location(String DatabaseURL, String DatabaseAlias, String SchemaName, String TableName, String UserName, String Password);

Creates an instance of the Location object with default attributes. Use the "set" methods described below to set the attributes. The meaning of each attribute is described in the corresponding "set" method.

get DatabaseURL();

Returns the string containing the database URL.

getDatabaseAlias();

Returns a string of type String that is the database alias for a location object.

getSchemaName();

Returns a string of type String that is the schema name for a location object.

getTableName();

Returns a string of type String that is the table name for a location object.

getUserName();

Returns a string of type String that is the user name for a location object.

getPassword();

Returns a string of type String that is the password for a location object.

setDatabaseURL(String URL);

Sets the Database URL in a Location object. The URL is used to create a JDBC connection to the database in which the table of interest resides or should be created.

setDatabaseAlias(String Alias);

Sets the Database Alias in a Location object. The alias identifies the database in which the table of interest resides or should be created.

setSchemaName(String SchemeName);

Sets the Schema Name in a Location object. The schema name identifies the schema for the data that is to be read or written.

setTableName(String TableName);

Sets the Table Name in a Location object. The table name identifies the input or output table used by a method.

setUserName(String UserName);

Sets the User Name in a Location object. The user name is the name to be used when connecting to the specified database.

setPassword(String Password);

Sets the password in a location object. The password is used in conjunction with the use name to connect to the database.

TuningSettings

Specifies settings to be applied when computing a recommendation. An instance of this class is passed to all recommendation requests.

Attributes

No public attributes.

Methods

There are two kinds of methods provided with this class:

A constructor that creates an TuningSettings instance

Methods that set attribute values

Methods that return attribute values

The following constructor creates a TuningSettings instance:

```
//Constructor
TuningSettings(DataSourceType dataSource
    InterestDimensionType interestDimension,
    PersonalizationIndexType   personalizationIndex ,
    ProfileDataBalanceType  profileDataBalance,
   ProfileUsageType  profileUsage);
```

Create a tuning settings instance for use in recommendation requests. For descriptions of the attributes, see the descriptions of the methods that set attribute values.

The following methods set attribute values:

setDataSourceType();

Sets the type of data to consider when computing recommendations. The options are

- Enum.DataSource.NAVIGATIONAL — use navigational (click stream) data only.
- Enum.DataSource.PURCHASING — use purchasing data only.
- Enum.DataSource.RATING — use rating data only.
- Enum.DataSource.DEMOGRAPHIC — use demographic data only.

- Enum.DataSource.ALL — use all data (navigational, purchasing, rating, and demographics).

setInterestDimension();

Sets the interest dimension that items should be ranked against. The options are

- Enum.InterestDimension.RATING — rank items according to the expected rating that the customer would assign to them. This interest dimension is useful in applications that collect rating data on items (that is, explicit preference or interest).
- Enum.InterestDimension.PURCHASING — rank items according to the likelihood that the customer will buy them. This interest dimension is useful in situations where selling items is the main goal.
- Enum.InterestDimension.NAVIGATION — rank items according to the likelihood that the customer would be interested in the items. This interest dimension is useful in situations where the intent is measured indirectly using navigational data (implicit preferences or interest). For example, how interested a customer would be in an article about a given subject.

setPersonalizationIndex();

Sets the attribute that specifies how "unusual" the recommendations returned will be. The options are

- Enum.PersonalizationIndex.LOW — recommend obvious items for a given customer profile; "Best Sellers" will appear more frequently among the recommendations in this case
- Enum.PersonalizationIndex.MEDIUM — recommend a balanced mixture of obvious and less obvious items for a given customer profile
- Enum.PersonalizationIndex.HIGH — recommend less obvious items for a given customer profile

setProfileDataBalance();

Sets the attribute that indicates whether to take data from current session or from history when making recommendations. The only valid option for bulk recommendations is:

- Enum.ProfileDataBalance.HISTORY — use historical data only for creating recommendation

setProfileUsage();

Sets the attribute that specifies if the recommendation list can include items in a customer's profile. Options are

- Enum.ProfileUsage.INCLUDE — include items in the profile
- Enum.ProfileUsage.EXCLUDE — do not include items in the profile

There are times when it is appropriate to exclude items in the profile; for example, you might not want to recommend that a customer buy books that he has already purchased.

The following methods return attribute values:

getDataSourceType();

Returns the type of data to consider when computing recommendations. The options are

- Enum.DataSource.NAVIGATIONAL — use navigational (click stream) data only.
- Enum.DataSource.PURCHASING — use purchasing data only.
- Enum.DataSource.RATING — use rating data only.
- Enum.DataSource.DEMOGRAPHIC — use demographic data only.
- Enum.DataSource.ALL — use all data (navigational, purchasing, rating, and demographics).

getInterestDimension();

Returns the interest dimension that items should be ranked against. The options are

- Enum.InterestDimension.RATING — rank items according to the expected rating that the customer would assign to them. This interest dimension is useful in applications that collect rating data on items (that is, explicit preference or interest).
- Enum.InterestDimension.PURCHASING — rank items according to the likelihood that the customer will buy them. This interest dimension is useful in situations where selling items is the main goal.
- Enum.InterestDimension.NAVIGATION — rank items according to the likelihood that the customer would be interested in the items. This interest dimension is useful in situations where the intent is measured indirectly using navigational data (implicit preferences or interest). For example, how interested a customer would be in an article about a given subject.

getPersonalizationIndex();

Returns the attribute that specifies how "unusual" the recommendations returned will be. The options are

- Enum.PersonalizationIndex.LOW — recommend obvious items for a given customer profile; "Best Sellers" will appear more frequently among the recommendations in this case
- Enum.PersonalizationIndex.MEDIUM — recommend a balanced mixture of obvious and less obvious items for a given customer profile
- Enum.PersonalizationIndex.HIGH — recommend less obvious items for a given customer profile

getProfileDataBalance();

Returns the attribute that indicates whether to take data from current session or from history when making recommendations. The options are

- Enum.ProfileDataBalance.HISTORY — use historical data only for creating recommendation
- Enum.ProfileDataBalance.BALANCED — use a balanced mixture of historical data and current session data for creating recommendations
- Enum.ProfileDataBalance.CURRENT — use current session data only for creating recommendation

getProfileUsage();

Returns the attribute that specifies if the recommendation list can include items in a customer's profile. Options are

- Enum.ProfileUsage.INCLUDE — include items in the profile
- Enum.ProfileUsage.EXCLUDE — do not include items in the profile

There are times when it is appropriate to exclude items in the profile; for example, you might not want to recommend that a customer buy books that he has already purchased.

3

Using the Recommendation Engine Batch Proxy

This chapter consists of an overview of the class and methods that are used to manage the recommendation engine proxy, to collect data, and to obtain recommendations, followed by the individual methods listed in alphabetical order. The supporting classes for these methods are described in Chapter 2.

All methods described in this chapter are public.

Overview

The recommendation proxy (REBatchProxy) methods can be divided according to function, as follows:

- Proxy creation and management, including customer profile management (load and purge customer profiles)
- Recommendation methods (obtain recommendations)

For examples of how to use these classes and methods, see Appendix A.

Location of Classes

To use the REProxyBATCH (and its exceptions), you must include the following statements in your Java program:

```
import oracle.dmt.op.re.reapi.batch.*;
import oracle.dmt.op.re.reexception.*;
```

These classes are installed on the system where Oracle9*i*AS is installed.

Proxy Creation and Management

The REProxyBatch.java class establishes the JDBC connection to the RE schema where the methods execute. The connection continues to exist until the connection is explicitly destroyed with the destroy() method. The class also includes customer profile management methods.

Customer Profile Management

You must load customer profiles from the MTR to the RE before you can request recommendations; after you are done, you should purge the loaded profiles from the RE. The methods are

- LoadCustomerProfiles();
- PurgeCustomerProfiles();

Recommendations

The following methods obtain recommendations:

- crossSellForItems
- rateItem
- recommendTopItems

Communicating the returned recommendations to the end user is the responsibility of the application. The recommendations are written to an output table; the schema of the output table depends on the method called. For details, see the description of each method.

Ratings in OP

Ratings in OP are in "ascending order of goodness", that is, the higher the rating, the more the user prefers the item. Low rated items are items that the user does not prefer. OP algorithms use these assumptions, so it is important that ratings are in ascending order of goodness.

Meaning of Returned Value for Recommendations

The meaning of the value returned for recommendation instances where ItemDetailData.attribute is equal to Enum.RecommendationAttribute.PREDICTION depends on the value of interestDimension as follows:

- For InterestDimension.RATING, the expected rating for the item is returned.

- For InterestDimension.PURCHASING or InterestDimension.NAVIGATION, a scaled probability is returned. The most probable item is assigned a value of 1 and other items are assigned values less than 1 that are proportional to how probable the items are compared to the most probable item.

Rules and Recommendations

OP uses rule tables stored in the RE to generate the recommendations requested by the recommendation methods. The rule tables are created when a package is built and stored in the RE when the package is deployed. The specific rule table used depends upon the RE Batch API call made. In general, the antecedents of the rules are matched against the data in cache (historical data only for RE Batch) and the probabilities of the various consequents are computed. These items are then ordered by probability, and numberOfItems (an API argument) items are returned.

Class and Method Details

The rest of this chapter contains detailed descriptions of the methods and classes, which are listed in alphabetical order.

crossSellForItems

Returns cross-sell recommendations for list of items in a table. The recommendations are written to a table.

Syntax

```
crossSellForItems (Location oItemIdsTableLocations,
   int iNumberofItems,
   TuningSettings oTuningSettings,
   FilteringSettings oFilteringSettings,
   Location oResultLocation);
```

Arguments

oItemIdsTable
Type Location, specifies the location of the table containing the list of items for which cross-sell recommendations are required. The table must be created before the method is invoked. The table must have the following schema:

```
ITEM_ID NUMBER
ITEM_TYPE VARCHAR2(30)
```

iNumberofItems
Type int, the number of items for which cross-sell recommendations are required.

oTuningSettings
Specifies data to use (purchasing or navigation) and personalization index (low, medium, or high). Interest dimension must be the same as the data source type of the input item

oFilteringSettings
Specifies taxonomyID, category filtering on or off, a category list, category membership (in, exclude leaves, exclude nodes, level, subtree nodes, or subtree leaves) and attribute filtering on or off.

oResultLocation
Type Location, specifies the location of the table containing the recommendations. The output table has the following schema:

```
SOURCE_ITEM_ID NUMBER
SOURCE_ITEM_TYPE VARCHAR2(30
REC_ITEM_ID NUMBER
```

REC_ITEM_TYPE VARCHAR2(30)
CONFIDENCE NUMBER

Return Value

None.

Usage Notes

Interest dimension must be the same as that of the data source type of the input item.

Data source type must be either navigational or purchasing. No other types are supported.

rateItem

Returns the rating, computed along an interest dimension, for an item. The rating could indicate, for example, the likelihood that customer X will buy item Y.

Syntax

```
rateItem(String sCustomerProfileTableName,
   Item  oItem,
   TuningSettings oTuningSettings,
   FilteringSettings oFilteringSettings,
   Location oResultLocation);
```

Arguments

sCustomerProfileTableName
Type String, specifies the name of the customer profiles table that has been previously loaded into the RE via LoadCustomerProfiles().

oItem
Type Item, the item type and item ID of the item to be rated.

oTuningSettings
Specifies settings to be applied when computing a recommendation; for details, see "TuningSettings" in Chapter 2

oFilteringSettings
Specifies taxonomyID, category filtering on or off, a category list, category membership (in, exclude leaves, exclude nodes, level, subtree nodes, or subtree leaves) and attribute filtering on or off.

oResultLocation
Type Location, specifies the location of the table containing the recommendations. The output table has the following schema:

```
CUSTOMER_ID VARCHAR2(32)
ITEM_ID VARCHAR2(30)
ITEM_TYPE NUMBER
PREDICTION NUMBER
```

Return Value

None.

Usage Notes

Interest dimension must be that same as that of the data source type.

In the tuning settings, the profile data balance is always historical, regardless of the value specified. only historical data is available for bulk recommendations; there is no session data.

How to interpret the recommendations depends on the value of interest dimension:

- For InterestDimension.RATING, return value is the expected rating
- For any other value of interest dimension, return value is a scaled probability with the most probable item assigned a value of 1.

For more information, see "Meaning of Returned Value for Recommendations" earlier in this chapter.

Exceptions

recommendTopItems

Returns the rating and other relevant item information for the numberOfItems items with the highest rating along the specified interest dimension. It answers questions such as: Which are the N items that person X is most likely to buy/like?

Syntax

```
recommendTopItems(String sCustomerProfileTableName,
   int iNumberOfTopItems,
   TuningSettings oTuningSettings,
   FilteringSettings oFilteringSettings,
   Location oResultLocation);
```

Arguments

sCustomerProfileTableName
Type String, specifies the name of the customer profiles table that has been previously loaded into the RE via LoadCustomerProfiles().

iNumberOfTopItems
Type int, the number of items to be recommended for each customer. This number represents the maximum number of items to be returned; the actual number returned may be less.

oTuningSettings
Specifies data to use (demographic, purchasing, rating, navigation, or all), interest dimension (interest, like, or buy) and personalization index (low, medium, or high)

oFilteringSettings
Specifies taxonomyID, category filtering on or off, a category list, category membership (in, exclude leaves, exclude nodes, level, subtree nodes, or subtree leaves) and attribute filtering on or off.

oResultLocation
Type Location, specifies the location of the table containing the recommendations. The output table has the following schema:

```
CUSTOMER_ID VARCHAR2(32)
ITEM_ID VARCHAR2(30)
ITEM_TYPE NUMBER
PREDICTION NUMBER
```

Return Value

None.

Exceptions

SQLException

IOException

CMException

Usage Notes

Interest dimension must be that same as that of the data source type.

Data source type must be either navigational or purchasing. No other types are supported.

In the tuning settings, the profile data balance is always historical, regardless of the value specified. only historical data is available for bulk recommendations; there is no session data.

How to interpret the recommendations depends on the value of interest dimension:

- For InterestDimension.RATING, return value is the expected rating
- For any other value of interest dimension, return value is a scaled probability with the most probable item assigned a value of 1.

For more information, see "Meaning of Returned Value for Recommendations" earlier in this chapter.

REProxyBatch.java

This class manages the database connection and customer profiles.

Attribute

String Name

Methods

```
//Constructor
REProxyBatch (String sProxyName,
   String sDbURL,
   String sUserName,
   String sPassword);
```

- sProxyName is the name of the object to be returned.
- sDbURL is a URL that points to the database where the RE to be used for recommendations resides.
- sUserName is the username for the RE schema
- sPassword is the password for the RE schema

Returns an REProxyBatch object with a valid JDBC connection to the specified RE engine.

```
void destroy();
```

Destroys the connection created with the constructor.

```
void loadCustomerProfiles(Location oCustomerIdsTable,
   String sCustomerProfileTableName);
```

- oLocation of type Location describes the database location of the customer IDs table.
- sCustomerProfileTableName is a string containing the name of the table to be created in the RE.

Load the MTR profiles into the RE for the customers in the customer IDs table. The schema of the customers ID table is

```
CUSTOMER_ID VARCHAR2(32)
```

The customer profile table is created in the RE schema to which REProxyBatch is connected. The format of the customer profiles table is

```
CUSTOMER_ID VARCHAR2(32)
```

ITEM_ID NUMBER
ITEM_TYPE VARCHAR2(30)
ATTRIBUTE_ID NUMBER
BIN_VALUE NUMBER
DATA_SOURCE_TYPE NUMBER(3)

void purgeCustomerProfiles(String sCustomerProfileTableName);

- sCustomerProfileTableName is the name of the RE table to be dropped.

Drops the specified table from the RE.

A

REProxyBatch API Examples and Usage

This appendix provides examples of REProxyBatch API use. In some instances, we provide coding skeletons; in others, we describe an approach for solving certain kinds of problems using OP.

REProxyBatch API Basic Usage

The REBatchProxy methods described in Chapter 3 permit you instrument your web site.

> **Note:** The RE Batch API classes are installed on the system where Oracle9*i*AS is installed. The tables that they use are installed on a different system (the system where Oracle9*i* is installed.) The following steps must be performed on the correct system.

To use REProxyBatch API calls, you must perform the following steps:

1. Create and deploy a package to the RE that you will use for recommendations.
2. Create an instance of REBatchProxy.
3. Create any required tables. (Alternatively, you can create the tables using SQL before you execute the program.)
4. Load customer profiles.
5. Execute the desired recommendation methods.
6. Purge the customer profiles that you loaded in step 4.
7. Destroy the database connection that you created in step 2.

You will now have a table containing the recommendations that you requested. You can use SQL to examine the table.

Code Sample: Recommend Top

The following code sample illustrates obtaining a recommendation.

```
// Create an instance of REProxyBatch

// Create customer table

// Load customer profiles

// Execute recommend_top

// Purge customer profiles loaded above

// Destroy the database connection held by REProxyBatch
```

Code Sample: Recommend Cross Sell

The following code sample illustrates obtaining cross-sell recommendations.

```
// Create an instance of REProxyBatch

// Create Items table

// Execute cross sell for items

// Destroy the database connection held by REProxyBatch
```

Recommendation Engine Usage

REBatchProxy requires at least one recommendation engine (RE) in at least one recommendation engine farm.

We recommend that the REs used for bulk recommendations not be used for any other purpose.

Note: If you try to deploy a package an RE while a batch program is running, the deployment will fail.

In general, you may want to use more than one RE to get satisfactory recommendation performance. Most applications will use multiple REs on different machines and subsequently different database instances.

Typically, for a given application, these REs will belong to the same RE farm. If a physical system has multiple processors, and the processors can be leveraged effectively by the database, the number of REs required for a given number of users can be reduced, perhaps even to one. See the administration guide for more information.

If your application has more than one RE available for use, it must determine which one to use. You can load different sets of customer profiles into different REs, generate appropriate recommendations, and them merge the recommendation tables, if desired.

Handling Multiple Currencies

OP stores currency data in the demographic table (for example, someone's income) as numbers; that is, OP does not store any kind of label. Both ten dollars (US) and ten pounds sterling (UK) are stored as "10".

There are several ways to ensure that currency data is interpreted correctly; the solution that you pick for your application depends on how your application uses currency data.

- Include a country code in customer demographics.

  This solution allows the country to be taken into account, but it does not closely associate the value with the country.

- Convert all currencies to a common currency such as Euros or United States dollars.

  This solution permits you to compare individual currency values in a meaningful way (10 pounds sterling is more than $10 US) but does not permit you preserve the difference between data such as a salary of $30,000 US in the US, versus the same $30,000 US salary in Brazil. You need such information if, for example, you want to recommend items to highly remunerated individuals in both the US and Brazil; the salary in US dollars of highly remunerated individuals will vary considerably from country to country.

  This approach requires that you preprocess the data outside of OP before OP creates recommendations.

- Bin currency values according to the mean to get relative values that can be compared across countries.

  This solution would permit you, for example, to determine the highly remunerated individuals for a given country, but it requires that you determine and maintain the bin boundaries appropriately.

  This approach requires that you preprocess the data outside of OP before OP creates recommendations.

Using Demographic Data

The schema of the MTR_CUSTOMER table consists of 50 generic attributes that can be mapped to any column in the site database. In order to support all different data types, all attributes are of type VARCHAR. Therefore, the mapped columns should be converted to strings. In this release of OP, these mapped columns are treated as categorical or numeric only. If any of the mapped columns is a DATE attribute, it should be converted to a number using the TO_NUMBER function. The converted values can then be binned just like any other attribute by specifying the bin boundaries.

There is binning for demographic data. The attributes that are binned can be of type boolean. In OP, the bin numbers are represented internally as integers, but the actual values are passed back to the calling applications. That is, the web application passes in the actual values and gets back actual values.

Handling Time-Based Items

For certain items, such as airline tickets, the price depends on when the item is purchased. For example, an airline ticket for a Boston to London flight has one price if it purchased 6 months before the date of the flight and a different price if it is purchased two days before the date of the flight.

If the web application assigns the same item ID to all tickets for the same trip, regardless of when they are purchased, then the items should have different item types, such as "6 month advance", "2 day advance", etc. Alternatively, the application could define taxonomies on the items and get recommendations on the categories.

If the application assigns different item IDs to the same flight purchased at different times (so that a ticket purchased 6 months before the flight has a different ID from a ticket for the same flight purchased 2 days before the flight), all tickets can have the same item type. In this case recommending item IDs may not be appropriate; therefore, the application should define a taxonomy and request recommendations on the categories.

B

REProxyBatchAPI Example

The sample program for RE Proxy Batch consists of a Java program and a property file. The sample program, property file, and the tables required to run it are installed when you install OP.

Sample Program Overview

The Java program **REBatchTest.java** and the property file **batchtest.txt** are in the `TBS` directory on the system where you have installed OP.

REBatchTest.java REProxyBatch allows you to execute a subset of recommendation functions in bulk. (REProxyRT scores one user/item at a time.) REProxyBatch reads a list of items/customers to be scored from an input table and writes the result to a new output table. This program reads its input from the property file **batchtest.ini**.

Sample Program Output

The input item details (for `rateItem` and `crossSellForItem`) are derived from the OP demo data. But in OP, the model built on the same data is not guaranteed to produce the same rules each time that it is run.Therefore, it is possible that the item being rated cannot be rated with the current set of rules. The output tables will either be empty (zero rows) or will contain fewer than expected records (if only some of the items are valid cross sell candidates etc.).

Executing the Sample Program

Follow these steps to execute the sample program:

1. Install OP.

2. The code and data for the sample program is installed into the following directories when you install OP:
   - The following code is installed in `${ORACLE_HOME}/dmt/reapi/batch/`
     - `batchtest.txt`
     - `README.txt`
     - `REBatchTest.java`
   - The following items associated with the data used by the sample program are installed in `${ORACLE_HOME}/dmt/reapi/batch/sampleData`
     - `create_batch_demo_input_tables.sql`
     - `customer_list_in.ctl`
     - `customer_list_in.txt`
     - `item_list_in.ctl`
     - `item_list_in.txt`
     - `load_batch_demo_data.sh`
3. Run the shell script `load_batch_demo_data.sh` to load the following tables:
   - `customer_list_in` — Used for `loadCustomerProfile`. (The output of `loadCustomerProfile` is used by `recommendTopItems` and `rateItem`.)
   - `item_list_in` — Used by `crossSellForItem`.
4. Compile the sample code. Your CLASSPATH variable should include the following zip / jar files:
   - `${ORACLE_HOME}/dmt/opreapi-batch.jar`
   - `${ORACLE_HOME}/dmt/oputil.jar`

   It also needs to include JDBC related jar / zip files:
   - `${ORACLE_HOME}/jdbc/lib/classes12.zip`
5. Change the property file to point to the appropriate entities. The comments in the property file and the file `README.txt` describes the exact changes that must be made.
6. Run `REBatchTest`, with the property file name as an input parameter.

RE Batch Sample Program

This section contains the code for the sample program and its property file.

batchtest.txt

The properties file for the sample program follows. Note that you must replace RE details and input/output table details to reflect your installation.

```
###
### Input file for REProxyBatch sample program
### Before Running, you will need to replace the following dummy strings with actual information:
### 1. RE* details ( Url,Username,Password) to point to the RE.
### 2. Input and Output (Result) table details for each of the calls.

#A unique name for proxy
ProxyName=REB_1

#Recommendation Engine details
REUrl=jdbc:oracle:thin:@myDBUrl
REUsername=REUser
REPassword=REPassword

#Input customer table location
Input.Url=jdbc:oracle:thin:@myDBUrl
Input.Alias=myDBAlias
Input.Schema=User1
Input.Table=customer_list_in
Input.Username=User1
Input.Password=Password1

#Customer profile table
# This table is created in RE by loadCustomerProfile. Once created
# it is used for recommendTopItems and rateItem
CustProfile=MY_CUSTOMER_PROFILE

#
# Details for recommendTopItems
#
# Number of items to be recommended per customer
TopN.NumberOfItems=10
#TuningSettings details
#valid DataSourceTypes are ALL, DEMOGRAPHIC, PURCHASING, RATING, NAVIGATION
TopN.DataSourceType=ALL
#valid InterestDimension: PURCHASING, RATING, NAVIGATION
```

```
TopN.InterestDimension=PURCHASING
#valid PersonalizationIndex: LOW, MEDIUM, HIGH
TopN.PersonalizationIndex=MEDIUM
## ProfileDataBalance needs to be specified as part of the TuningSettings object
## but its value is not used by REProxyBatch
#valid ProfileDataBalance: HISTORY, CURRENT, BALANCED
TopN.ProfileDataBalance=HISTORY
#valid ProfileUsage:INCLUDE, EXCLUDE
TopN.ProfileUsage=INCLUDE
# FilteringSettings details
TopN.Taxonomy=1
#Category list is a series of numbers separated by "-"
TopN.CategoryList=1-2-3-4-5
#Valid CategoryMembership: ExcludeItems, ExcludeCategories, IncludeItems, IncludeCategories,
#       level, SubTreeItems, SubTreeCategories, AllItems, AllCategories
TopN.CategoryMember=AllItems
# Result table details
TopNResult.Url=jdbc:oracle:thin:@myDBUrl
TopNResult.Alias=myDBAlias
TopNResult.Schema=User2
TopNResult.Table=TopN_RESULTS
TopNResult.Username=User2
TopNResult.Password=Password2

#
# Details for rateItem
#
#TuningSettings details
RateI.ItemID=417
RateI.ItemType=MOVIE
RateI.DataSourceType=RATING
RateI.InterestDimension=RATING
RateI.PersonalizationIndex=LOW
## ProfileDataBalance needs to be specified as part of the TuningSettings object
## but its value is not used by REProxyBatch
RateI.ProfileDataBalance=HISTORY
RateI.ProfileUsage=INCLUDE
RateI.Taxonomy=1
# Result table details
RateIResult.Url=jdbc:oracle:thin:@myDBUrl
RateIResult.Alias=myDBAlias
RateIResult.Schema=User3
RateIResult.Table=RATEITEM_RESULTS
RateIResult.Username=User3
RateIResult.Password=Password3
```

```
#
# Details for crossSellForItem
#
#Input items table details
ItemTable.Url=jdbc:oracle:thin:@myDBUrl
ItemTable.Alias=myDBAlias
ItemTable.Schema=User4
ItemTable.Table=item_list_in
ItemTable.Username=User4
ItemTable.Password=User4
# Number of items to be recommended per input item
XSell.NumberOfItems=10
#TuningSettings details
XSell.DataSourceType=NAVIGATION
XSell.InterestDimension=NAVIGATION
XSell.PersonalizationIndex=HIGH
## ProfileDataBalance needs to be specified as part of the TuningSettings object
## but its value is not used by REProxyBatch
XSell.ProfileDataBalance=HISTORY
XSell.ProfileUsage=EXCLUDE
#FilteringSettings details
XSell.Taxonomy=1
XSell.CategoryList=1-3-5-7-9
XSell.CategoryMember=AllItems
# Result table details
XSellResult.Url=jdbc:oracle:thin:@myDBUrl
XSellResult.Alias=myDBAlas
XSellResult.Schema=User4
XSellResult.Table=XSELL_RESULTS
XSellResult.Username=User5
XSellResult.Password=Password5
```

REBatchTest.java

The sample program follows. Note that you must replace RE details and input/output table details to reflect your installation.

```
###
### Input file for REProxyBatch sample program
### Before Running, you will need to replace the following dummy strings with actual information:
### 1. RE* details ( Url,Username,Password) to point to the RE.
### 2. Input and Output (Result) table details for each of the calls.

#A unique name for proxy
```

```
ProxyName=REB_1

#Recommendation Engine details
REUrl=jdbc:oracle:thin:@myDBUrl
REUsername=REUser
REPassword=REPassword

#Input customer table location
Input.Url=jdbc:oracle:thin:@myDBUrl
Input.Alias=myDBAlias
Input.Schema=User1
Input.Table=customer_list_in
Input.Username=User1
Input.Password=Password1

#Customer profile table
# This table is created in RE by loadCustomerProfile. Once created
# it is used for recommendTopItems and rateItem
CustProfile=MY_CUSTOMER_PROFILE

#
# Details for recommendTopItems
#
# Number of items to be recommended per customer
TopN.NumberOfItems=10
#TuningSettings details
#valid DataSourceTypes are ALL, DEMOGRAPHIC, PURCHASING, RATING, NAVIGATION
TopN.DataSourceType=ALL
#valid InterestDimension: PURCHASING, RATING, NAVIGATION
TopN.InterestDimension=PURCHASING
#valid PersonalizationIndex: LOW, MEDIUM, HIGH
TopN.PersonalizationIndex=MEDIUM
## ProfileDataBalance needs to be specified as part of the TuningSettings object
## but its value is not used by REProxyBatch
#valid ProfileDataBalance: HISTORY, CURRENT, BALANCED
TopN.ProfileDataBalance=HISTORY
#valid ProfileUsage:INCLUDE, EXCLUDE
TopN.ProfileUsage=INCLUDE
# FilteringSettings details
TopN.Taxonomy=1
#Category list is a series of numbers separated by "-"
TopN.CategoryList=1-2-3-4-5
#Valid CategoryMembership: ExcludeItems, ExcludeCategories, IncludeItems, IncludeCategories,
#       level, SubTreeItems, SubTreeCategories, AllItems, AllCategories
TopN.CategoryMember=AllItems
```

```
# Result table details
TopNResult.Url=jdbc:oracle:thin:@myDBUrl
TopNResult.Alias=myDBAlias
TopNResult.Schema=User2
TopNResult.Table=TopN_RESULTS
TopNResult.Username=User2
TopNResult.Password=Password2

#
# Details for rateItem
#
#TuningSettings details
RateI.ItemID=417
RateI.ItemType=MOVIE
RateI.DataSourceType=RATING
RateI.InterestDimension=RATING
RateI.PersonalizationIndex=LOW
## ProfileDataBalance needs to be specified as part of the TuningSettings object
## but its value is not used by REProxyBatch
RateI.ProfileDataBalance=HISTORY
RateI.ProfileUsage=INCLUDE
RateI.Taxonomy=1
# Result table details
RateIResult.Url=jdbc:oracle:thin:@myDBUrl
RateIResult.Alias=myDBAlias
RateIResult.Schema=User3
RateIResult.Table=RATEITEM_RESULTS
RateIResult.Username=User3
RateIResult.Password=Password3

#
# Details for crossSellForItem
#
#Input items table details
ItemTable.Url=jdbc:oracle:thin:@myDBUrl
ItemTable.Alias=myDBAlias
ItemTable.Schema=User4
ItemTable.Table=item_list_in
ItemTable.Username=User4
ItemTable.Password=User4
# Number of items to be recommended per input item
XSell.NumberOfItems=10
#TuningSettings details
XSell.DataSourceType=NAVIGATION
XSell.InterestDimension=NAVIGATION
```

```
XSell.PersonalizationIndex=HIGH
## ProfileDataBalance needs to be specified as part of the TuningSettings object
## but its value is not used by REProxyBatch
XSell.ProfileDataBalance=HISTORY
XSell.ProfileUsage=EXCLUDE
#FilteringSettings details
XSell.Taxonomy=1
XSell.CategoryList=1-3-5-7-9
XSell.CategoryMember=AllItems
# Result table details
XSellResult.Url=jdbc:oracle:thin:@myDBUrl
XSellResult.Alias=myDBAlas
XSellResult.Schema=User4
XSellResult.Table=XSELL_RESULTS
XSellResult.Username=User5
XSellResult.Password=Password5
```