Develop an IMS Application using Java and Open Database

Skill Level: Beginner to Intermediate

Poonam Chitale (pchitale@us.ibm.com)
Software Engineer, IBM®

Joshua Newell (newelljo@us.ibm.com)
Software Engineer, IBM®

IBM Corporation
August 2013

Abstract
This tutorial takes you through the steps of using Rational® Developer for System z® Version 8.0.3 to write a Java™ application to access IMS databases through the IMS Universal DL/I driver and the IMS Universal JDBC driver.

About this tutorial
This tutorial will take you through the steps of writing a Java application to access IMS databases using the IMS Universal DL/I driver and the IMS Universal JDBC driver.

Customers who store business data in IMS databases want an easy way to access their data. They also want to be able to develop applications for IMS using modern and standardized programming solutions. The IMS Universal drivers, part of the IMS Version 12 Open Database solution, are software components that provide Java applications with connectivity to IMS databases from z/OS® and from distributed environments through TCP/IP.

The IMS Universal drivers are built on industry standards and open specifications. Java applications that use the IMS Universal drivers can reside on the same logical partition (LPAR) or on a different LPAR from the IMS subsystem. Two types of connectivity are supported by the IMS Universal drivers: local connectivity to IMS databases on the same LPAR (type-2 connectivity) and distributed connectivity through TCP/IP (type-4 connectivity).

This tutorial will help to familiarize you with using two of the IMS Universal drivers:

- IMS Universal DL/I driver, which provides a stand-alone Java application programming interface (API) for writing granular queries to IMS databases using programming semantics similar to traditional IMS DL/I calls
- IMS Universal JDBC driver, which provides a stand-alone Java Database Connectivity (JDBC) 3.0 driver for making structured query language (SQL)-based database calls to IMS databases.

In this tutorial, you will run Java applications in a Windows® environment and connect to the IMS database using type-4 connectivity mode.
Distributed and local connectivity with the IMS Universal drivers

The IMS Universal drivers support distributed (type-4) and local (type-2) connectivity to IMS databases. The connectivity type is specified in the driverType connection property. In this tutorial exercise, you will use type-4 connectivity.

- **Type-4 connectivity:** With type-4 connectivity, the IMS Universal drivers can run on any platform that supports TCP/IP and a Java Virtual Machine (JVM), including z/OS. To access IMS databases using type-4 connectivity, the IMS Universal drivers first establish a TCP/IP-based socket connection to IMS Connect. IMS Connect is responsible for routing the request to the IMS databases using the Open Database Manager (ODBM), and sending the response back to the client application. The DRDA® protocol is used internally in the implementation of the IMS Universal drivers. You do not need to know DRDA to use the IMS Universal drivers.

![Figure 1: Distributed (type-4) connectivity](image)

- **Type-2 connectivity:** Local (or type-2) connectivity with the IMS Universal drivers is targeted for the z/OS platform and runtime environments. You would use type-2 connectivity when connecting to IMS subsystems in the same logical partition (LPAR). In this tutorial, you will not need type-2 connectivity.

Objectives

To understand and gain hands-on experience creating Java applications to access and manipulate enterprise data residing on the IMS database.

Upon completion of this study, you will be able to perform these tasks:

- Create a Java application to access IMS data by issuing IMS DL/I calls to the IMS database through the IMS Universal DL/I driver
- Create a Java application to access IMS data by issuing SQL calls to the IMS database through the IMS Universal JDBC driver
• Deploy and run a Java application in a Windows environment

**System requirements for the tutorial:**

Software installed on Windows
- Rational Developer for System z Version 8.0.3 (RDz)
- IMS Universal drivers libraries
  - imsudb.jar
- Sample Java project
  - IMSDBJavaApplicationLab.zip

System software installed on IBM z/OS
- IMS Version 12 configured with Open Database Manager (ODBM)
- IMS Connect Version 12

**Checklist for first-time implementation**

You may find it helpful have the following information and resources ready before proceeding with your first implementation of the Java applications using the IMS Universal drivers. The information and resources to run this tutorial is provided in the checklist below.

**Table 1: Implementation checklist**

<table>
<thead>
<tr>
<th>Information or resource</th>
<th>Your environment</th>
<th>For this tutorial</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMS Connect host name (or IP address) and DRDA port number</td>
<td>Obtain this information Host name: ZSERVEROS.DEMOS.IBM.COM from IMS system programmers.</td>
<td>DRDA port number: 7001</td>
</tr>
<tr>
<td>IMS data store name (IMS ID)</td>
<td>Obtain this information Datastore name: IMSD from IMS system programmers.</td>
<td></td>
</tr>
<tr>
<td>z/OS user ID and password</td>
<td>Obtain this information Userid: EM4ZIMS from IMS system programmers.</td>
<td></td>
</tr>
<tr>
<td>MetadataURL to the Java metadata file generated by the IMS Explorer</td>
<td>Obtain this information MetadataURL:(DatabaseName) from IMS application programmers.</td>
<td></td>
</tr>
<tr>
<td>Workspace directory and project name to be used when generating artifacts</td>
<td>A naming standard is recommended.</td>
<td>Workspace directory: C:satshare\imsjavalab\workspace</td>
</tr>
</tbody>
</table>
Overview of development tasks

To complete this tutorial, you will perform the following tasks:

1 **Task 1 - Install the tutorial sample project**
   1.1 Import the IMSDBJavaApplicationLab sample project
   1.2 Verify that the IMS Universal drivers library is located on the build path

2 **Task 2 – Access IMS data with the IMS Universal DL/I driver**
   2.1 Connect to the IMS database through the IMS Universal DL/I driver
      2.1.1 Open the DLIAPIAssigment.java sample application
      2.1.2 Set the connection properties
   2.2 Issue DL/I calls to access the IMS database
      2.2.1 Exercise 1 - Retrieve data in an IMS database
      2.2.2 Exercise 2: Retrieve batch data in an IMS database
      2.2.3 Exercise 3: Create SSALists with multiple segments, specify qualifications, and mark specific fields for retrieval
      2.2.4 Exercise 4: Utilize command codes for DL/I

3 **Task 3 - Access IMS data with the IMS Universal JDBC driver**
   3.1 Connect to the IMS database through the IMS Universal JDBC driver
      3.1.1 Open the JDBCApiAssignment.java sample application
      3.1.2 Set the connection properties
   3.2 Issue SQL calls to access the IMS database
      3.2.1 Exercise 1 - Retrieve all fields of a segment
      3.2.2 Exercise 2 - Retrieve fields of a segment based on a conditional statement
      3.2.3 Exercise 3 - Order SQL query output by field values
      3.2.4 Exercise 4 - Retrieve a specific field of a segment
      3.2.5 Exercise 5 – Retrieve multiple fields from multiple segments
1 Task 1 - Install the tutorial sample project
In this task, you will import the tutorial sample project to IBM Rational Developer for System z (RDz), and verify that the Java library with the code to run the IMS Universal DL/I and JDBC drivers is installed.

1.1 Switch to the Java perspective
Switch from the default z/OS Projects perspective to the Java perspective.

1. IBM Rational Developer for System z V 8.0.3 is started and you are using the 
   C:\share\imsjavlab\workspace as your workspace directory.

Important:
For this tutorial, you will use C:\share\imsjavlab\workspace as your workspace directory.

The Workspace
In RDz, a workspace is a directory that stores files for your projects. You can select your own directory or take the default directory. Artifacts created by RDz will be stored in this directory.

2. From the menu bar, select Window > Open Perspective > Other.

Figure 2: Opening a perspective in Rational Developer for System z

3. Scroll down and select Java from the Open Perspective dialog box.

Figure 3: Choosing the Java perspective
4. Press **OK** to switch to the Java perspective.
5. To verify that you are in Java perspective, make sure that the Java button appears in the upper right corner of RDz, as shown in the figure below.

![Figure 4: Verifying that the Java perspective is opened.](image)

**What is a perspective?**
A perspective defines the initial set and layout of views in the Workbench window. Within the window, each perspective shares the same set of editors. Each perspective provides a set of functionality aimed at accomplishing a specific type of task or works with specific types of resources. For example, the Java perspective combines views that you would commonly use while editing Java source files, while the Debug perspective contains the views that you would use while debugging Java programs.

### 1.2 Import the IMSDBJavaApplicationLab sample project
Import the files for the IMSDBJavaApplicationLab sample project into the RDz workspace.

**The IMSDBJavaApplicationLab sample project**
The sample project includes the Java library that contains the IMS Universal drivers required for this tutorial. The sample also includes sample Java application code that you will customize to connect to an IMS database and issue database access calls.

1. From the menu bar, click **File > Import** to open the Import dialog box.
2. From the Import dialog box, select **General > Existing Projects into Workspace** and click **Next**.

3. From the Import Projects page, select **Select root directory** and click **Browse**.
4. Browse to the directory `C:\Lab Files\IMS Java Application Development\IMSDBJavaApplicationLab` and click **OK**.

---

**Figure 5: Launching the Import dialog box**

**Figure 6: Launching the Import Existing Projects into Workspace wizard**
5. Make sure that the checkbox for **IMSDBJavaApplicationLab** is selected and click **Finish**.

The sample project IMSDBJavaApplicationLab should appear in the Package Explorer view.

---

**Figure 7: Importing a project from an archive file**

![Import Projects dialog box]

**Figure 8: Package Explorer view after successfully importing the sample project**

![Package Explorer view]
1.3 Verify that the IMS Universal drivers library is located on the build path

Verify that the Java archive file imsudb.jar is correctly located in the build path of this project.

1. Right click on the project in the Package Explorer view and select Build Path > Configure Build Path

Figure 9: Opening the Java Build Path properties page

2. From the Java Build Path page, click on the Libraries tab. Verify that the file `imsudb.jar` – `IMSDBJavaApplicationLab` is present.

The `imsudb.jar` library
The `imsudb.jar` file contains the Java classes, interfaces, and metadata required to use the IMS Universal DL/I driver and the IMS Universal JDBC driver.
3. Click **OK** to save your changes and exit the Java Build Path page.
2 Task 2 – Access IMS data with the IMS Universal DL/I driver

In this task, you will write a Java application to connect to an IMS database and manipulate data using a DL/I-based syntax with the IMS Universal DL/I driver.

What is DL/I?
Data Language/I (DL/I) is the IMS data manipulation language, which is a common high-level interface between a user application and IMS. DL/I calls are invoked from application programs written in languages such as Java, PL/I, COBOL, VS Pascal, C, and Ada. It also can be invoked from assembler language application programs by subroutine calls. IMS lets the user define data structures, relate structures to the application, load structures, and reorganize structures.

By using the IMS Universal DL/I driver, you can build segment search arguments (SSAs) and use the methods of the program communication block (PCB) object to read, insert, update, delete, or perform batch operations on segments. You can gain full navigation control in the segment hierarchy.

Basic programming model for a Java application using the IMS Universal DL/I driver
In general, to write a IMS Universal DL/I driver application, follow these steps:

1. Import the com.ibm.ims.dli package that contains the IMS Universal DL/I driver classes, interfaces, and methods.
2. Connect to an IMS database subsystem.
3. Obtain a program specification block (PSB), which contains one or more PCBs.
4. Obtain a PCB handle, which defines an application’s view of an IMS database and provides the ability to issue database calls to retrieve, insert, update, and delete database information.
5. Obtain an unqualified segment search argument list (SSAList) of one or more segments in the database hierarchy.
6. Add qualification statements to specify the segments targeted by DL/I calls.
7. If retrieving data, mark the segment fields to be returned.
8. Execute DL/I calls to the IMS database.
9. Handle errors that are returned from the DL/I programming interface.
10. Disconnect from the IMS database subsystem.

2.1 Connect to the IMS database through the IMS Universal DL/I driver
Before you can execute DL/I calls from your IMS Universal DL/I driver application, you must connect to an IMS database.

Develop an IMS application using Java and OpenDB
© Copyright IBM Corporation 2011, 2012. All rights reserved.
2.1.1 Open the DLIApiAssignment.java sample application

The DLIApiAssignment.java sample application
This sample application contains skeleton Java code for connecting to the IMS database and issuing DL/I data access calls using the IMS Universal DL/I driver.

1. From the Package Explorer view, expand IMSDBJavaApplicationLab > src > com.ibm.ims.db.exercise.

Figure 11: Navigating to the DLIApiAssignment.java sample application

2. From the Package Explorer view, double click on the file DLIApiAssignment.java to open the sample application in the Java editor.

Maximizing a view is the ability to increase a view to the maximum possible size on the screen. This can be accomplished by double-clicking on the view tab. To go back to the original view size, double-click on the view tab again.
Develop an IMS application using Java and OpenDB
© Copyright IBM Corporation 2011, 2012. All rights reserved.
13 of 41

The Java editor
The Java editor provides specialized features for editing Java code. The editor includes support for syntax highlighting, content/code assist, code formatting, import assistance, and integrated debugging features.

2.1.2 Set the connection properties
1. In the Java editor, scroll down the application source code until you find the Java main method shown in the screenshot below.

Figure 12: The opened DLIApiAssignment.java sample application in the Java editor

Figure 13: Java main method in the DLIApiAssignment sample application

Moving your cursor to a specific line number
RDz provides a shortcut to move your cursor directly to a specific line in an editor. To go to a specific line, press Ctrl + L from the editor. Enter the line number and press OK.
Develop an IMS application using Java and OpenDB

Displaying line numbers in the editor
Line numbers can be displayed directly in the editor by going to Windows > Preferences. In the Preferences Dialog Menu navigate to General > Editors > Text Editors and check the box next to Show line numbers.

Figure 14: Configuring the editor to display line numbers

2. In line 44 of the code, delete the constant REPLACE_THIS and replace it with IMSConnectionSpec.DRIVER_TYPE_4 to set the driver connectivity type.

Java code assist
RDz provides code assist for Java applications. By pressing CTRL + space, the Java editor will display a list of possible commands variables for that line. Try specifying the driver type by typing IMS and pressing CTRL + space and scrolling to the constant IMSConnectionSpec. Alternatively, when you type a period (.) after a class, the Java code assist displays a menu of methods and variables that the class can invoke. Try it after IMSConnectionSpec and select DRIVER_TYPE_4.

3. In line 56 of the code, delete the string "your.host.name.com" and replace it with "zserveros.demos.ibm.com" to set the host.

The datastoreServer property
The host variable in the sample application is used to set the datastoreServer property. This connection property contains the name or IP address of the data store server (IMS Connect). You can provide either the host name (for example, dev123.svl.ibm.com) or the IP address (for example, 192.166.0.2). In this tutorial, the target IMS Connect has already been pre-configured for you. Use zserveros.demos.ibm.com as the dataStoreServer.
4. In line 57 of the code, delete the string "" and replace it with "IMSD" to set the datastoreName.

**The datastoreName property**
This connection property contains the name of the IMS data store to access. When using type-4 connectivity, the datastoreName property must match either the name of the data store defined to Open Database Manager (ODBM) or be blank. In this tutorial, the target IMS data store has already been created for you and pre-populated with data. Use IMSD as the datastoreName.

5. In line 58 of the code, delete the string "yourID" and replace it with "EM4ZIMS" to set the username.

**The user and password properties**
The user and password connection properties are the user name and password used for the connection to IMS Connect. This information can typically be obtained from your RACF® administrator.

6. In line 60 of the code, verify that the drdaPort value 7001 has already been set for you.

**The portNumber property**
The drdaPort variable in the sample application is used to set the portNumber property. This connection property is the TCP/IP server port number to be used to communicate with IMS Connect. The portNumber property is not required when using type-2 connectivity. In this tutorial, the target IMS Connect has already been pre-configured for you. Use 7001 as the drdaPort.

7. In line 82 of the code, delete the constant REPLACE_THIS and replace it with the string "class://com.ibm.ims.db.databaseviews.DFSSAM09DatabaseView" to set the metadataURL.

**The metadataURL property**
This connection property is the location of the database metadata representing the target IMS database. The metadataURL property is the fully qualified name of the Java metadata class generated by the IMS Enterprise Suite DLIModel utility plug-in, based on the PSB and DBD source files of the target IMS database. The Java metadata class must be generated before coding a Java application to access the target IMS database using the IMS Universal drivers. The format of the metadataURL is: "class://packageName.className"

In this tutorial, the Java metadata class has already been generated for you. Use class://com.ibm.ims.db.databaseviews.DFSSAM09DatabaseView as the metadataURL.
8. Press Ctrl + S to save your code changes.

After completing this step, your Java application should be ready to connect using the Universal DL/I driver. Next, you will need to modify the Java application code to issue data access calls to IMS.
The Parts Order sample database

This tutorial uses the Parts Order database that is provided in the IMS Installation Verification Program (IVP). You can refer to this diagram when working on the exercises in this tutorial.

The diagram below shows the hierarchical structure of the segments in the Parts Order database. Each rectangle represents a database segment. PARTROOT is the root segment of this database, and STANINFO and STOKSTAT are its child segments. STOKSTAT has CYCCOUNT and BACKORDR as its child segments. Each segment contains one or more fields that contain data. For example, PARTKEY is a field in the PARTROOT segment.

Figure 17: Segments of the Parts Order database (reference only)
2.2 Issue DL/I calls to access the IMS database

The following exercises in this section will show you how to issue DL/I calls in your Java application to retrieve data from the IMS database using the IMS Universal DL/I driver.

### Lab exercises

This task contains several programming exercises for you to complete. These exercises will help to familiarize you with basic data access operations using the IMS Universal DL/I driver. At certain points indicated in the instructions, you will be asked to provide the correct code. For your reference, we have provided the exercise solutions. You can find the code with the exercise solutions from the Package Explorer view by opening IMSDBJavaApplicationLab > com.ibm.ims.db.exercise.solution > DLIApiAssignment.java

### 2.2.1 Exercise 1 - Retrieve data in an IMS database

In this exercise, you will retrieve data in an IMS database by issuing DL/I Get Unique and Get Next calls through the IMS Universal DL/I driver.

#### Using the Get Unique (GU) and Get Next (GN) DL/I calls

If an input message contains more than one segment, a Get Unique call retrieves the first segment of the message and Get Next (GN) calls retrieve the remaining segments.

When issued from the IMS Universal DL/I driver, the Get Unique call retrieves a specific segment or collection of segments on a hierarchic path from an IMS database. The GU call also establishes the position in the database from which additional segments can be processed in a forward direction.

The Get Next call retrieves the next segment or collection of segments on a hierarchic path from an IMS database. The GN call usually proceeds forward along the hierarchy of a database from the current database position to the next required segment. To modify the GN call to start at an earlier position than the current position in the database, you can use an IMS command code. The Get Next call returns a Path object representing the hierarchic path from the root segment to the segment the cursor is currently positioned on. The Path object includes the data stored in the segments along the hierarchic path.

Exercise 1 begins on line 103 of the DLIApiAssignment.java sample application, where the function displayPARTROOT(psb) is invoked. To go to the start of the function, go to line 107, move the mouse over the displayPARTROOT(psb) function invocation, and press F3.

Figure 18: Navigating to the start of Exercise 1

```java
// Exercise 1: Issue Get Unique and Get Next calls.
// Uses getUnique and getNext calls.
System.out.println("Starting Exercise 1:");
displayPARTROOT(psb);
System.out.println("Completed Exercise 1\n\n");
```
2.2.1.1 Exercise 1 - Step 1: Define an unqualified SSAList to specify the segments to retrieve

1. In line 146 of the code, delete the constant `REPLACE_THIS` and replace it with the Java code statement to get an unqualified SSAList for the PARTROOT segment. You can find the answer after Figure 19 below.

Hint:
Use the PCB object that has been created (`partspcb1`) to call the `getSSAList(String)` method. Pass in the segment name ("PARTROOT") as the input parameter.

The SSAList interface

The `com.ibm.ims.dli.SSAList` interface represents a list of segment search arguments (SSAs) used to specify the segments to target in a particular database call. Use the SSAList interface to construct each segment search argument in the list, and to set the command codes and lock class for the segment search arguments. Each SSA in the SSAList can be qualified or unqualified. A SSA qualification can be used to filter the segments to update or retrieve on a hierarchic path.

Figure 19: Defining the unqualified segment search argument list

Verify your Java code statement:
In line 146, your Java code statement should look like this:

```
SSAList ssaList = partspcb1.getSSAList("PARTROOT");
```

2.2.1.2 Exercise 1 - Step 2: Issue a Get Unique DL/I call to retrieve segments

1. In line 164 of the code, delete the constant `REPLACE_THIS` and replace it with the Java code statement to issue a Get Unique DL/I call. You can find the answer after figure 20 below.

Hint:
Use the PCB object that was previously created (`partspcb1`) to call the `getUnique(Path, SSAList, boolean)` method.
- Pass in the Path object that was previously created (`path`) as the 1st input parameter.
- Pass in the SSAList object that was previously created (`ssalList`) as the 2nd input parameter.
- Pass in the boolean value `false` as the 3rd input parameter. False indicates that this DL/I call is not a Get Hold Unique call.
Figure 20: Insert code to issue the Get Unique call

```java
// TODO - Make a Get Unique call using partspcb1
// The call takes 3 parameters: Path, SSAList, and a boolean
// indicating if this is a HOLD call (GH1, GHN, GHNP).
// The call will return true if the call is successful or false if
// not.
if (REPLACE_THIS) {
  // Format the output
  System.out.println("PARTKEY\t\tPART\t\tPARTDESC");
  System.out.println("\n----------------");
}
```

Verify your Java code statement:
In line 164, your Java code statement should look like this:

```java
if (partspcb1.getUnique(path, ssaList, false)) {
```

2.2.1.3 Exercise 1 – Step 3: Issue a Get Next DL/I call

1. In line 178 of the code, delete the constant REPLACE_THIS and replace it with the Java code statement to issue a Get Next DL/I call. You can find the answer after Figure 21 below.

Hint:
Use the PCB object that was previously created (partspcb1) to call the `getNext(Path, SSAList, boolean)` method.
- Pass in the Path object that was previously created (path) as the 1st input parameter
- Pass in the SSAList object that was previously created (ssaList) as the 2nd input parameter
- Pass in the boolean value false as the 3rd input parameter, to indicate that this DL/I call is not a Get Hold Next call.

Figure 21: Insert code to issue a Get Next call

```java
while (REPLACE_THIS) {
  System.out.println(path.getString("PARTKEY").trim()
               + "\t" + path.getString("PARTDESC").trim)
}
```

Verify your Java code statement:
In line 178, your Java code statement should look like this:

```java
while (partspcb1.getNext(path, ssaList, false)) {
```
2.2.1.4 Exercise 1 – Step 4: Run the application and verify the output results

1. Press Ctrl + S to save your changes to the files.
2. Right click on the Java editor and select Run As > Java Application.

Figure 22: Running the Java application

3. An Errors in Workspace dialog box will appear but you can safely ignore it. Click on Proceed to continue.
4. In the Console view, verify that the output results look like the screenshots below.
2.2.2 Exercise 2: Retrieve batch data in an IMS database

In this exercise, you will retrieve batch data from an IMS database by issuing a Batch Retrieve call through the IMS Universal DL/I driver.

Batch Retrieve

You can use the batch retrieve call to retrieve multiple segments from an IMS database in a single call. Instead of a client application making multiple GU and GN calls, IMS performs all the GU and GN processing and returns the results back to the client in a single batch network operation. The fetch size property determines how much data is returned on each batch network operation.

Exercise 2 begins on line 110 of the DLIApiAssignment.java sample application. At the beginning of Exercise 2, the code for this exercise has been commented out.

2.2.2.1 Exercise 2 – Step 1: Uncomment the code for Exercise 2

1. In the Java editor, highlight lines 112 to 114 of the DLIApiAssignment.java sample application and press Ctrl + / to uncomment the code.

Figure 25: Code for Exercise 2 (before uncomment)
Code comments allows comment statements that will not be compiled and executed to be inserted directly into the application source code. In RDz, blocks of code can be commented and uncommented by highlighting that block and pressing Ctrl + /.

Figure 26: Code for Exercise 2 (after uncomment)

```java
// Exercise 2: Use batchRetrieve for the DL/I API.
// Uses batchRetrieve calls.
System.out.println("Starting Exercise 2: ");
displayPARTROOTUsingBatchRetrieve(psb);
System.out.println("Completed Exercise 2 
");
```

2. The function displayPARTROOTUsingBatchRetrieve contains the Java code for the batch retrieval operation. In line 113, move your mouse over the displayPARTROOTUsingBatchRetrieve(psb) function invocation and press F3 to open the function declaration.

2.2.2.2 Exercise 2 – Step 2: Issue a Batch Retrieve call to retrieve multiple segments

1. In line 210 of the code, delete the constant REPLACE_THIS and replace it with the Java code statement to issue a Batch Retrieve call. You can find the answer after the figure below.

**Hint:**
Use the PCB object that was previously created (partspcb1) to call the batchRetrive(SSAList) method.
- Pass in the SSAList object that was previously created (ssaList) as the input parameter.

Figure 27: Insert code to issue a batch retrieve call

```java
// TODO – Make a Batch Retrieve call
// Using the pcb, you can issue a batchRet
// which will return a PathSet or a set of
// Each path in the set represents data fr
// segment or row in the database.
PathSet ps = REPLACE_THIS;
```

**Verify your Java code statement:**
In line 210, your Java code statement should look like this:

```java
PathSet ps = partspcb1.batchRetrieve(ssaList);
```
2.2.2.3 Exercise 2 – Step 3: Commit the unit of work

1. In line 227 of the code, delete the constant `REPLACE_THIS` and replace it with the Java code statement to commit your unit of work. You can find the answer after the figure below.

**Hint:**
Use the PSB object was previously created (`psb`) to call the `commit()` method.

![Figure 28: Insert code to issue a commit call](image)

`psb.commit();`

2.2.2.4 Exercise 2 – Step 4: Run the application and verify the output results

1. Press Ctrl + S to save your changes to the files.
2. Right click on the Java editor and select Run As section > Java Application, as shown in section 2.2.1.4.
3. An Errors in Workspace dialog box will appear. Click on Proceed.
4. In the Console view, verify that the output results look like the screenshots below.
2.2.3 Exercise 3: Create SSALists with multiple segments, specify qualifications, and mark specific fields for retrieval

In this exercise, you will mark specific segment fields for retrieval from the IMS database. You will also specify the number of rows of data for the IMS Universal DL/I driver to retrieve.

Marking segment fields for retrieval with the IMS Universal DL/I driver

In your Java application, you can specify which segment fields are to be returned from a database retrieve call by using the markFieldForRetrieval or the markAllFieldsForRetrieval methods. Following the IMS default, all of the fields in the lowest level segment specified by the SSAList are initially marked for retrieval.

The markFieldForRetrieval method

This SSAList method is used to mark a specific field for retrieval from the database. The markFieldForRetrieval method is used together with getPathForRetrieveReplace() and with the data retrieval methods in the PCB interface. When a retrieve call is made, the resulting Path object will contain all the fields that have been marked for retrieval.

The markAllFieldsForRetrieval method

This method is used to mark all fields in the specified segment for retrieval from the database. The markAllFieldsForRetrieval method is used together with getPathForRetrieveReplace() and with the data retrieval methods in the PCB interface. When a retrieve call is made the resulting Path object will contain only the fields marked for retrieval. Following the IMS default, all of the fields in the lowest level segment specified by the SSAList are initially marked for retrieval.

Exercise 3 begins on line 116 of the DLIApiAssignment.java sample application. At the beginning of Exercise 3, the code for this exercise has been commented out.
2.2.3.1 Exercise 3 – Step 1: Uncomment the code for Exercise 3

1. In the Java editor, highlight lines 118 to 120 of the DLIApiAssignment.java sample application and press Ctrl + / to uncomment the code.

Figure 31: Code for Exercise 3 (before uncomment)

```java
// Exercise 3: Creating SSA Lists with multiple qualifications, and marking specific fields for
// System.out.println("Starting Exercise 3:");
// displayBACKORDR(psb);
// System.out.println("Completed Exercise 3\n\n");
```

Figure 32: Code for Exercise 3 (after uncomment)

```java
// Exercise 3: Creating SSA Lists with multiple qualifications, and marking specific fields:
System.out.println("Starting Exercise 3:");
displayBACKORDR(psb);
System.out.println("Completed Exercise 3\n\n");
```

2. The function `displayBACKORDR` contains the Java code for the retrieval. In line 119, move your mouse over the `displayBACKORDR(psb)` function invocation and press F3 to open the function declaration.

2.2.3.2 Exercise 3 – Step 2: Build an unqualified SSAList

1. In line 246 of the code, delete the constant `REPLACE_THIS` and replace it with the Java code statement to build an unqualified SSAList for a hierarchic path of segments ranging from the top-level PARTROOT segment to the bottom-level BACKORDR segment. You can find the answer after the figure below.

**Hint:**
Declare a new SSAList variable (ssaList). Use the PCB object that has been created (partspcb1) to call the `getSSAList(String, String)` method.
- Pass in the PARTROOT segment name ("PARTROOT") as the 1st input parameter
- Pass in the BACKORDR segment name ("BACKORDR") as the 2nd input parameter

Figure 33: Insert code to build the unqualified SSAList

```java
// TODO - Use the pcb to build an:
// containing a range of segments:
// PARTROOT
// STORSTATS
// BACKORDR
//
// REPLACE_THIS;
```
2.2.3.3 Exercise 3 - Step 3: Mark the fields to retrieve

1. In line 262 of the code, delete the constant REPLACE_THIS and replace it with the Java code statement to mark the WORKORDER field for retrieval from the BACKORDR segment. In line 263, add the Java code statement to mark the ORDERQTY field for retrieval from the same segment. You can find the answer after the figure below.

Hint:
Use the SSAList object that has been created (ssaList) to call the method.
- Pass in the segment name ("BACKORDR") as the 1st input parameter, to indicate the name of the segment in the SSAList containing the field
- Pass in the field name ("WORKORDER") as the 2nd input parameter, to indicate the name of the field to be marked for retrieval from the database
- Pass in the boolean value true as the 3rd input parameter, to indicate that this field should be retrieved from the database

In the next line, create a similar statement to mark the ORDERQTY field for retrieval.

Figure 34: Insert code to mark the segment fields to retrieve

```
// TODO - Only retrieve specific fields
// Now we only want two fields to be returned
// --> WORKORDER and ORDERQTY
//
// Use the markFieldForRetrieval method.
ssaList.REPLACE_THIS;
```

Verify your Java code statement:
In line 262 and 263, your Java code statements should look like this:

```
ssaList.markFieldForRetrieval("BACKORDR", "WORKORDER", true);
ssaList.markFieldForRetrieval("BACKORDR", "ORDERQTY", true);
```

2.2.3.4 Exercise 3 – Step 4: Specify the number of rows to fetch per network call

1. In line 274 of the code, delete the constant REPLACE_THIS and replace it with the Java code statement to set the fetch size property to 30. You can find the answer after the figure below.

Hint:
Use the PCB object that has been created (partspcb1) to call the method.
- Set the number of rows to fetch (30) as the 1st input parameter
Fetch size property

The fetch size is the number of rows physically retrieved from the IMS database per network call. A list of rows is represented by a Path instance containing one or more segments that match the segment search argument criteria specified by an SSAList. This is set for you internally. You can also set the fetch size using the setFetchSize method from the PCB interface. Setting the fetch size allows a single request to return multiple rows at a time, so that each application request to retrieve the next row does not always result in a network request.

Figure 35: Insert code to change the fetch size

```java
// TODO - Using the PCB specify how many rows
// Set the fetch size for IMS to send back
// This helps to cut down on the number of
partspcb1.setFetchSize(30);
```

Verify your Java code statement:
In line 274, your Java code statement should look like this:
```
partspcb1.setFetchSize(30);
```

2.2.3.5 Exercise 3 – Step 5: Print the retrieved segment fields from the path

1. In line 287 of the code, delete the 1st instance of the constant REPLACE_THIS and change the System.out.println statement to print the value of the WORKORDER field returned by IMS. In the same line, delete the 2nd instance of the constant REPLACE_THIS and change the Java code statement to print the value of the ORDERQTY field returned by IMS. You can find the answer after the figure below.

Hint:
Use the Path object that has been created (path) to call the getString(string) method.
- Set the 1st parameter to the field name ("WORKORDER"), to retrieve the value of this field.
Use a similar method call to retrieve the value of the ORDERQTY field.

Figure 36: Modifying the System.out.println statement to print the retrieved segment fields

```java
// TODO - Print WORKORDER and ORDERQTY from the path
// \
// \'t\' represents a tab
// ','. is used to concatenate string values
System.out.println(REPLACE_THIS + "\t\t" + REPLACE_THIS);
```
2.2.3.6 Exercise 3 – Step 6: Run the application and verify the output results

1. Press Ctrl + S to save your changes to the files.
2. Right click on the Java editor and select Run As > Java Application, as shown in section 2.2.1.4.
3. An Errors in Workspace dialog box will appear. Click on Proceed.
4. In the Console view, verify that the output results look like the screenshots below.

Figure 37: Exercise 3 result output in the Console view

<table>
<thead>
<tr>
<th>Problem</th>
<th>Javadoc</th>
<th>Declaration</th>
<th>Console</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting Exercise 3:</td>
<td>WORKORDER</td>
<td>ORDERQTY</td>
<td></td>
</tr>
<tr>
<td>OPR1932</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPR14937</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPR15309</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPR15309</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPR16956</td>
<td>50</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Completed Exercise 3

2.2.4 Exercise 4: Utilize command codes for DL/I

In this exercise, you will add a command code in the SSAList to retrieve a sequence of segments.

Command codes for DL/I

SSAs can also include one or more command codes, which can change and extend the functions of DL/I calls. For example, you can use the D command code to retrieve or insert a sequence of segments in a hierarchic path with one call rather than retrieving or inserting each segment with a separate call. A call that uses the D command code is called a path call.

Exercise 4 begins on line 122 of the DLIApiAssignment.java sample application. At the beginning of Exercise 4, the code for this exercise has been commented out.

2.2.4.1 Exercise 4 – Step 1: Uncomment the code for Exercise 4

1. In the Java editor, highlight lines 123 to 125 of the DLIApiAssignment.java sample application and press Ctrl + / to uncomment the code.

Figure 38: Code for Exercise 4 (before uncomment)

```java
// Exercise 4: Utilizing IMS Command Codes in your
// System.out.println("Starting Exercise 4:");
// displayPARITROOTandBACKORDERData(psb);
// System.out.println("Completed Exercise 4\n\n");
```
2. The function `displayPARTROOTandBACKORDER` contains the code for the batch retrieval operation. In line 124, move your mouse over the `displayPARTROOTandBACKORDERData(psb)` function invocation and press F3 to open the function declaration.

2.2.4.2 Exercise 4 – Step 2: Add a command code to the SSAList

1. In line 333 of the code, delete the constant `REPLACE_THIS` and replace it with the Java code statement to add the D command code. You can find the answer after the figure below.

Hint:
Use the SSAList object that has been created (`ssaList`) to call the `addCommandCode(String, byte)` method.
- Set the name of the segment ("PARTROOT") as the 1st input parameter
- Set the command code (SSAList.CC_D) as the 2nd input parameter

**Figure 40: Insert the code to add an IMS command code**

```java
// TODO - Add an IMS command code to the SS.
// We want to retrieve data from both PART
// order to do this in IMS you need to add
// to the PARTROOT because the way the SSAL
// you will only get data from the leaf seg
//
// After adding the command code D the SSA:
// PARTROOT =D(PARTKEY = 027618032P101 )
// STICKSTAT
// BACKORDR
//
// Use the addCommandCode method.

REPLACE_THIS
```

Verify your Java code statement:
In line 333, your Java code statement should look like this:

```java
ssaList.addCommandCode("PARTROOT", SSAList.CC_D);
```
2.2.4.3 Exercise 4 – Step 3: Run the application and verify the output results

1. Press Ctrl + S to save your changes to the files.
2. Right click on the Java editor and select Run As > Java Application, as shown in section 2.2.1.4.
3. In the Console view, verify that the output results look like the screenshots below.

Figure 41: Exercise 4 result output in the Console view

<table>
<thead>
<tr>
<th>PARTKEY</th>
<th>PARDESC</th>
<th>WORKORDER</th>
<th>ORDERQTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>027618032P101</td>
<td>CAPACITOR</td>
<td>0FR14932</td>
<td>10</td>
</tr>
<tr>
<td>027618032P101</td>
<td>CAPACITOR</td>
<td>0FR14937</td>
<td>10</td>
</tr>
<tr>
<td>027618032P101</td>
<td>CAPACITOR</td>
<td>0FR15309</td>
<td>10</td>
</tr>
<tr>
<td>027618032P101</td>
<td>CAPACITOR</td>
<td>0FR18309</td>
<td>10</td>
</tr>
<tr>
<td>027618032P101</td>
<td>CAPACITOR</td>
<td>0FR16956</td>
<td>50</td>
</tr>
</tbody>
</table>

Completed Exercise 4
3 Task 3 - Access IMS data with the IMS Universal JDBC driver

In this task, you will write a Java application to connect to an IMS database and manipulate data using structured query language (SQL) with the IMS Universal JDBC driver.

What is JDBC?

Java Database Connectivity (JDBC) is an application programming interface (API) that Java applications use to access relational databases or tabular data sources. The JDBC API is the industry standard for database-independent connectivity between the Java programming language and any database that has implemented the JDBC interface. The client uses the interface to query and update data in a database.

IMS support for JDBC lets you write Java applications that can issue dynamic SQL calls to access IMS data and process the result set that is returned in tabular format. The IMS Universal JDBC driver is designed to support a subset of the SQL syntax with functionality that is limited to what the IMS database management system can process natively. Its DBMS-centric design allows the IMS Universal JDBC driver to fully leverage the high performance capabilities of IMS. The IMS Universal JDBC driver also provides aggregate function support, and ORDER BY and GROUP BY support.

Basic programming model for a Java application using the IMS Universal JDBC driver

The IMS Universal JDBC driver supports the standard programming model for using JDBC drivers. For more information about the JDBC programming model, see the JDBC Basics tutorial by SUN.

3.1 Connect to the IMS database through the IMS Universal JDBC driver

Before you can execute SQL calls from your IMS Universal JDBC driver application, you must connect to an IMS database.

3.1.1 Open the JDBCApiAssignment.java sample application

The JDBCApiAssignment.java sample application

This sample application contains skeleton Java code for connecting to the IMS database and issuing SQL data access calls using the IMS Universal JDBC driver.

1. In the Package Explorer view, expand IMSDBJavaApplicationLab > src > com.ibm.ims.db.exercise
2. From the Package Explorer view, double click on `JDBCApiAssignment.java` to open the sample application in the Java editor.

3.1.2 Set the connection properties

1. In line 22 of the code, verify that this application will connect using type-4 connectivity.
2. In line 35, verify that `7001` is set as the IMS Connect DRDA port number.
3. In line 38, verify that `"class://com.ibm.ims.db.databaseviews.DFSSAM09DatabaseView"` is set as the metadataURL.
3.2 Issue SQL calls to access the IMS database

The following exercises in this section will show you how to issue SQL calls in your Java application to retrieve data from the IMS database using the IMS Universal JDBC driver.

<table>
<thead>
<tr>
<th>Hierarchical database elements in IMS</th>
<th>Equivalent relational database elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment name</td>
<td>Table name</td>
</tr>
<tr>
<td>Segment instance</td>
<td>Table row</td>
</tr>
<tr>
<td>Segment field name</td>
<td>Column name</td>
</tr>
<tr>
<td>Segment unique key</td>
<td>Table primary key</td>
</tr>
<tr>
<td>Virtual foreign key field</td>
<td>Table foreign key</td>
</tr>
</tbody>
</table>

3.2.1 Exercise 1 - Retrieve all fields of a segment

In this exercise, you will retrieve all the fields of a segment by issuing a SELECT statement using the IMS Universal JDBC driver.

Using the SELECT keyword

Use the SELECT statement to retrieve data from one or more tables. The result is returned in a tabular result set. The syntax for a simple SELECT query is:

```
SELECT column_name(s) FROM table_name
```

An asterisk * can be used in place of column_name to represent all columns of that table. Because IMS is a hierarchical database, column_name maps to field_name and table_name maps to segment_name.

When using the SELECT statement with the IMS Universal JDBC driver:

- If you are selecting from multiple tables and the same column name exists in one or more of these tables, you must table-qualify the column or an ambiguity error will occur.
- The FROM clause must list all the tables you are selecting data from. The tables listed in the FROM clause must be in the same hierarchic path in the IMS database.
- In Java applications using the IMS JDBC drivers, connections are made to PSBs. Because there are multiple database PCBs in a PSB, queries must specify which PCB in a PSB to use. To specify which PCB to use, always qualify segments that are referenced in the FROM clause of an SQL statement by prefixing the segment name with the PCB name. You can omit the PCB name only if the PSB contains only one PCB.

1. Move your cursor to line 20 of the sample application. You will modify this statement to issue different SQL queries in this task.
2. In line 20, construct a SQL query to retrieve all of the fields of the `PARTSPCB1.PARTROOT` segment. Note that the segment name must start with the PCB qualifier.
   - Set the query string to "SELECT * FROM PARTSPCB1.PARTROOT"

3. Press Ctrl + S to save your changes to the files.

4. Right click on the Java editor and select Run As > Java Application, as shown in section 2.2.1.4.

5. In the Console view, verify that the result output looks like the screenshots below.

Figure 45: Beginning of Task 3 - Exercise 1 result output

<table>
<thead>
<tr>
<th>PARTY</th>
<th>PARDESC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AN960C10</td>
<td>WASHER</td>
</tr>
<tr>
<td>CK05CM181K</td>
<td>CAPACITOR</td>
</tr>
<tr>
<td>C81G04K</td>
<td>MK1J0K</td>
</tr>
<tr>
<td>KJN97B</td>
<td>DIODE CODE-A</td>
</tr>
<tr>
<td>MS16995-28</td>
<td>SCREW</td>
</tr>
<tr>
<td>N1P5003F000</td>
<td>SCREW</td>
</tr>
<tr>
<td>RC070F273J</td>
<td>RESISTOR</td>
</tr>
<tr>
<td>10651293P009</td>
<td>RESISTOR</td>
</tr>
<tr>
<td>250236-001</td>
<td>CAPACITOR</td>
</tr>
<tr>
<td>250239</td>
<td>TRANSISTOR</td>
</tr>
<tr>
<td>250241-001</td>
<td>CONNECTOR</td>
</tr>
</tbody>
</table>

Figure 46: End of Task 3 - Exercise 1 result output

<table>
<thead>
<tr>
<th>PARTY</th>
<th>PARDESC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2903333-001</td>
<td>DISCRIMINATO</td>
</tr>
<tr>
<td>946325-086</td>
<td>FIN</td>
</tr>
<tr>
<td>950060-006</td>
<td>RELAY</td>
</tr>
<tr>
<td>954017-001</td>
<td>RESISTOR</td>
</tr>
<tr>
<td>958007-180</td>
<td>RESISTOR</td>
</tr>
<tr>
<td>96528-067</td>
<td>RESISTOR</td>
</tr>
<tr>
<td>966034-001</td>
<td>SOCKET</td>
</tr>
<tr>
<td>974810-010</td>
<td>THERMOSTAT</td>
</tr>
<tr>
<td>975105-001</td>
<td>TRANSFORMER</td>
</tr>
<tr>
<td>989036-001</td>
<td>TRANSFORMER</td>
</tr>
</tbody>
</table>
3.2.2 Exercise 2 - Retrieve fields of a segment based on a conditional statement

In this exercise, you will retrieve specific fields of a segment based on a conditional statement by issuing a SELECT statement with a WHERE clause using the IMS Universal JDBC driver.

Using the WHERE keyword

Use the WHERE keyword in SQL to select data conditionally. The syntax for a conditional select query is:

```
SELECT column_name(s) FROM table_name WHERE column_name operator value
```

Note that for text values, the value must be enclosed in quotes. Operators on text values perform binary comparisons. The IMS Universal JDBC driver converts the WHERE clause in an SQL query to a segment search argument (SSA) list when querying a database. SSA rules restrict the type of conditions you can specify in the WHERE clause.

1. In line 20, construct a SQL query that will display all fields of the PARTSPCB1.PARTROOT segment where the PARTKEY field is greater than '025'. Note that the PARTKEY field contains data that is alphanumeric.

   - Set the query string to

```
SELECT * FROM PARTSPCB1.PARTROOT WHERE PARTKEY > '025'
```

2. Press Ctrl + S to save your changes to the files.
3. Right click on the Java editor and select Run As > Java Application, as shown in section 2.2.1.4.
4. In the Console view, verify that the beginning of the result output looks like the screenshot below.

Figure 47: Beginning of Task 3 - Exercise 2 result output

```
PARTKEY   PARTDESC
0268354-016   89136-016   NAS671CL NUT
0268354-018   89136-018   7734304P8661TO RES
0268599-002   89136-002   WIRE WRAP
0268599-012   89136-125   TRANSFORMER
0268599-014   89136-145   CMOS3100K03
0268599-016   89136-165   CMOS3200U03
0268599-035   89136-355   CP03ALKE153K3 CAPAC
027606654P001   7060654P001   ELE TUBE
027606654P002   7060654P002   NUT
027606654P003   7060654P003   LAMP HOLDER
027618032P101   7060654P101   CAPACITOR
```
### 3.2.3  Exercise 3 - Order SQL query output by field values

In this exercise, you will retrieve data from a segment in sorted order issuing a SELECT statement with an ORDER BY clause using the IMS Universal JDBC driver.

**Using the ORDER BY keyword**

Use the **ORDER BY** clause in SQL to sort the results of a SQL query in ascending or descending order. The syntax for an ordered select query is:

```sql
SELECT column_name(s) FROM table_name ORDER BY column_name ASC|DESC
```

Note that **ASC** is used for ascending order and **DESC** is used for descending order. The field names that are specified in an ORDER BY clause must match exactly the field name that is specified in the SELECT statement.

1. In line 20, construct a SQL query that will retrieve all fields of the `PARTSPCB1.PARTROOT` segment and sort the results by the `PART` field in descending order.
   - Set the query string to "**SELECT * from PARTSPCB1.PARTROOT ORDER BY PART DESC**"
2. Press Ctrl + S to save your changes to the files.
3. Right click on the Java editor and select **Run As > Java Application**, as shown in section 2.2.1.4.
4. In the Console view, verify that the result output looks like the screenshots below.

**Figure 48: Beginning of the Task 3 - Exercise 3 result output**

```
PARTKEY  PARTID  PARTDESC
02RC07GR723J  RC07GR723J  RESISTOR
02MS1P000F000  N51P000F000  SCREW
02MS1GR98-25  MS1GR98-25  SCREW
02JZU0N9765  JZU0N9765  DIODE C0E-A
02CSR15G104ML  CSR15G104ML  RESISTOR
02CR05G181R  CR05G181R  CAPACITOR
02JZU160C10  JZU160C10  WASHER
02989036-001  989036-001  TRANSFORMER
02989105-001  989105-001  TRANSFORMER
02974810-010  974810-010  THERMOSTAT
02968349-001  968349-001  SOCKET
```

**Figure 49: End of the Task 3 - Exercise 3 result output**

```
02004456-004  004564  CIRCUIT
02252522-003  252522-003  COUPLING
02250591  250591  SERVO VALVE
02250796  250796  SWITCH
02250794  250794  RESISTOR
02250281-001  250281-001  CONNECTOR
02250299  250299  TRANSISTOR
02250236-001  250236-001  CAPACITOR
02106B1293F009  106B1293F009  RESISTOR
```
3.2.4 Exercise 4 - Retrieve a specific field of a segment

In this exercise, you will retrieve a specific field in a segment. By querying only specific fields instead of selecting all the fields in a particular segment, you can reduce network overhead when using the IMS Universal JDBC driver.

1. In line 20, construct a SQL query that will display only the PART field from the PARTSPCB1.PARTROOT segment.
   - Set the query string to "SELECT PART FROM PARTSPCB1.PARTROOT"
2. Press Ctrl + S to save your changes to the files.
3. Right click on the Java editor and select Run As > Java Application, as shown in section 2.2.1.4.
4. In the Console view, verify that the result output looks like the screenshots below.

Figure 50: Beginning of Task 3 - Exercise 4 result output

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PART</td>
<td>AN96C10</td>
</tr>
<tr>
<td></td>
<td>CK050W181K</td>
</tr>
<tr>
<td></td>
<td>CSR1G104K1</td>
</tr>
<tr>
<td></td>
<td>JDD19976B</td>
</tr>
<tr>
<td></td>
<td>MSL16995-28</td>
</tr>
<tr>
<td></td>
<td>NS1F3003F000</td>
</tr>
<tr>
<td></td>
<td>RC070F273J</td>
</tr>
<tr>
<td></td>
<td>105B1933F000</td>
</tr>
<tr>
<td></td>
<td>280236-001</td>
</tr>
<tr>
<td></td>
<td>250239</td>
</tr>
<tr>
<td></td>
<td>280241-001</td>
</tr>
</tbody>
</table>

Figure 51: End of Task 3 - Exercise 4 result output

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>550551-123</td>
<td>930333-001</td>
</tr>
<tr>
<td>946325-086</td>
<td>950060-006</td>
</tr>
<tr>
<td>945017-001</td>
<td>988007-180</td>
</tr>
<tr>
<td>960528-067</td>
<td>968534-001</td>
</tr>
<tr>
<td>974810-010</td>
<td>975105-001</td>
</tr>
<tr>
<td>989036-001</td>
<td></td>
</tr>
</tbody>
</table>

3.2.5 Exercise 5 – Retrieve multiple fields from multiple segments

In this exercise, you will issue a SELECT query to retrieve data from segments that are on the same and on different hierarchical paths in the IMS database. Note that the PARTROOT and the BACKORDR segments are on the same hierarchic path, while the CYCCOUNT segment is on a separate hierarchic path.

Develop an IMS application using Java and OpenDB
© Copyright IBM Corporation 2011, 2012. All rights reserved.
38 of 41
Retrieving fields from multiple segments

In SQL queries to relational databases, the JOIN keyword is typically used to query data from multiple tables based on a relationship between the tables. IMS does not support using the JOIN keyword explicitly, because IMS is a hierarchical database and it is possible that two segments are unrelated to each other. However, an implicit join will be performed if the segments fall within the same hierarchical path. The syntax for this is the same as for a SELECT query. Note that multiple column names and table names can be specified as long as a comma is used to separate them. IMS allows issuing a SELECT call to retrieve data from segments that are not on the same hierarchical path, if a logical relationship has been defined between them.

1. In line 20, construct a SQL query that will display the PART field from the PARTSPCB1.PARTROOT segment and the BACKKEY field from the PARTSPCB1.BACKORDR segment.
   - Set the query string to "SELECT PART, BACKKEY FROM PARTSPCB1.PARTROOT, PARTSPCB1.BACKORDR"
2. Press Ctrl + S to save your changes to the files.
3. Right click on the Java editor and select Run As > Java Application, as shown in section 2.2.1.4.
4. In the Console view, verify that the result output looks like the screenshot below.

Figure 52: Result output of Task 3 - Exercise 5 part 1

5. In line 20, construct a SQL query that will display the PHYSICALCOUNT field from the PARTSPCB1.CYCCOUNT segment and the WORKORDER field from the PARTSPCB1.BACKORDR segment.
   - Set the query string to "SELECT PHYSICALCOUNT, WORKORDER FROM PARTSPCB1.CYCCOUNT, PARTSPCB1.BACKORDR"
6. Press Ctrl + S to save your changes to the files.
7. Right click on the Java editor and select Run As > Java Application, as shown in section 2.2.1.4.
8. In the Console view, verify that the query fails with this error message: "The tables BACKORDR and CYCCOUNT specified in the query cannot be joined together. They are not along the same hierarchic path in the database".
Resources

Learn

- View the Information Management Software for z/OS Solutions Information Center for the latest information and educational resources available for Information Management System (IMS), including the Java API reference for the IMS Universal drivers.

- Visit the Rational software area on developerWorks for technical resources and best practices for Rational Software Delivery Platform products.

- Explore Rational computer-based, Web-based, and instructor-led online courses. Hone your skills and learn more about Rational tools with these courses, which range from introductory to advanced. The courses on this catalog are available for purchase through computer-based training or Web-based training. Additionally, some “Getting Started” courses are available free of charge.

- Subscribe to the Rational Edge e-zine for articles on the concepts behind effective software development.

- Check out the Information Management IMS zone on developerWorks.

- Subscribe to the IBM developerWorks newsletter, a weekly update on the best of developerWorks tutorials, articles, downloads, community activities, webcasts and events.

- Explore Rational computer-based, Web-based, and instructor-led online courses. Hone your skills and learn more about Rational tools with these courses, which range from introductory to advanced. The courses on this catalog are available for purchase through computer-based training or Web-based training. Additionally, some “Getting Started” courses are available free of charge.

- Subscribe to the IBM developerWorks newsletter, a weekly update on the best of developerWorks tutorials, articles, downloads, community activities, webcasts and events.

- Browse the technology bookstore for books on these and other technical topics.


- Find code sample for cast-to-DLI approach for JDBC here: https://www.ibm.com/developerworks/community/blogs/8a337b1c-3c0c-48a5-b7cc-7f805884dbb9/entry/new_example_available_cast_to_dli_from_the_ims_universaljdbc_driver?lang=en

Get products and technologies

- Download trial versions of IBM Rational software.

- Download a trial version of IBM Rational Developer for system z.

Discuss

- Participate in the discussion forum.

- Check out developerWorks blogs and get involved in the developerWorks community.
About the authors

Poonam Chitale is a Software Engineer for IMS Open Database solution
Joshua Newell is a Software Engineer for IMS Open Database solution and level 2 support

Trademark notice

IBM, the IBM logo and ibm.com are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.