# Executing a View Object Query in Code

**About executeQuery**

A statement like the following executes the View Object's current query statement and refreshes the collection of results (with on-demand fetching, as is the norm).

myVO.executeQuery();

If the existing state of the View Object includes pending changes (inserts, updates, deletes), they are cached in the underlying Entity Object caches and are not affected by refreshing the query. The changes are still pending and the inserted and updated rows will appear in their current pending state in the View Object.

The executeQuery method does not post, commit, or validate changes. Instead, call postChanges and validate explicitly, as needed. For example:

// Pre-validate all changes in this View Object  
 myVO.validate();

// or you can pre-validate all changes in the Application Module.   
 myVO.getApplicationModule().validate();

// Now post all changes   
 myVO.getApplicationModule().getTransaction().postChanges();

// Then re-execute the query   
 myVO.executeQuery();

If other database users add new rows to the database in the meantime, and they meet the WHERE criteria of your View Object, they will appear in the View Object after calling executeQuery. Methods such as setWhereClause that change the query clauses do not execute the query. You can change the query clauses as many times as you want. However, the new query does not take affect until you call executeQuery.

Data retrieved from the RowSet may not match the query clauses in the following scenario:

setWhereClause("a = 10") /\* line 1 \*/  
executeQuery()   
fetch data /\* line 3 \*/  
setWhereClause("a != 10")   
fetch more data /\* line 5 \*/

Rows returned from line 5 will still have a = 10. Call executeQuery after line 4 to get rows with a != 10.

If you do the following, line 6 will still show b = 11 (the pending, changed value set in line 4).

setWhereClause("a = 10") /\* line 1 \*/  
executeQuery()   
retrieve a row /\* line 3 \*/  
row.setAttribute("b", 11) /\* Say b was equal to 15 in the DB \*/   
executeQuery() /\* line 5 \*/  
fetch rows

In contrast, consider the following scenario, which changes an attribute in the WHERE clause:

setWhereClause("a = 10") /\* line 1 \*/  
executeQuery()   
retrieve a row /\* line 3 \*/  
row.setAttribute("a", 11) /\* NOTE we're setting "a" this time! \*/   
executeQuery() /\* line 5 \*/  
fetch rows

When you call executeQuery in line 5, changes to the row in line 4 have not been posted to the database. In the database, this row still has a = 10. So, when you fetch rows in line 6, the row in line 3 is retrieved because the database session still sees "a = 10" in the database. However, on the client, or internally in response to row.getAttribute("a"), it will return 11.

**About hasNext**

The hasNext method executes a View Object's query if it has not already been executed. It is useful for looping through a result set, as shown in the following code snippet.

vo.executeQuery();  
while (vo.hasNext()) {   
 Row row = vo.next(); // Get the next row in the result set.  
 System.out.println(row.getAttribute(1)); // Do something with the row.

}

**View Object Query Examples**

The following code example executes a query defined in a View Object created using wizards at design time, then scrolls through the result set. The example uses the generic Application Module and calls Row.getAttribute to access attributes.

package d2e;  
import oracle.jbo.\*;  
import javax.naming.Context;  
import javax.naming.InitialContext;  
import java.util.Hashtable;

public class QueryDemo  
{  
 public static void main(String[] args)  
 {  
 // Helper routine connects to the generic Application Module.  
 ApplicationModule appMod =  
 getGenericAppMod(JboContext.PLATFORM\_LOCAL);

// Specify the XML file that defines the View Object.  
 // Format: <package>.<filename>  
 String voDefFile = "d2e.EmpView";

// Identify the View Object. Must be a valid Java identifier.  
 String voName = "demoVO";

// Create the View Object within the context defined   
 // by the Application Module.  
 ViewObject vo = appMod.createViewObjectF(voName, voDefFile);  
 printViewObject(vo);  
 vo.remove();  
 }

public static ApplicationModule getGenericAppMod(String platform)  
 {  
 Hashtable env = new Hashtable(2);  
 env.put(Context.INITIAL\_CONTEXT\_FACTORY, JboContext.JBO\_CONTEXT\_FACTORY);  
 env.put(JboContext.DEPLOY\_PLATFORM, platform);  
 try  
 {  
 Context ic = new InitialContext(env);  
 String theAMDefName = "d2e.D2eModule";  
 ApplicationModuleHome home = (ApplicationModuleHome)ic.lookup(theAMDefName);  
 ApplicationModule appMod = home.create();  
 appMod.getTransaction().connect("jdbc:oracle:thin:scott/tiger@pc3:1521:ORCL");  
 return appMod;  
 }  
 catch(Exception e)  
 {  
 e.printStackTrace();  
 }  
 return null;  
 }

public static void printViewObject(ViewObject vo)  
 {  
 // Execute the query, print results to the screen.  
 vo.executeQuery();  
 while (vo.hasNext()) {  
 Row row = vo.next();  
 String rowDataStr = "";

// How many attributes (columns) is the View Object using?  
 int numAttrs = vo.getAttributeCount();

// Column numbers start with 0, not 1.  
 for (int columnNo = 0; columnNo < numAttrs; columnNo++) {

// See also Row.getAttribute(String name).  
 Object attrData = row.getAttribute(columnNo);  
 rowDataStr += (attrData + "\t");  
 }  
 System.out.println(rowDataStr);  
 }  
 }  
}