Achieve improved database interoperability with SQL and RDB aliases

Nathan Skalsky (skalsky@us.ibm.com)
Staff Software Engineer
IBM
18 September 2014

In today's complex environments, where data is spread across numerous homogeneous and heterogeneous databases, simplification is a necessity. Three-part naming offers that simplification for database connectivity by allowing skilled data architects, programmers, and system administrators the ability to easily manage remote Distributed Relational Database Architecture (DRDA) connections within SQL applications and data centers. This article gives you insight into the options available to use DRDA more proficiently through three-part naming and RDB aliases. Understanding these concepts can enable you to improve your SQL database interoperability in your SQL applications or daily use and also gain a level of database abstraction and portability.

Introduction

Remote three-part naming is nothing new and has been around since IBM i 7.1. More recently, IBM® DB2® for i three-part naming support for DRDA has become more exciting by allowing the direct use of a relational database (RDB) alias as the first part of a three-part name.

Three-part naming is the action of using a three-part name wherever allowed within a SQL statement. When a three-part name is used within an SQL statement, the initiator of the SQL statement is known as the application requestor (AR). The AR references distributed data over an implicit DRDA connection made to an application server (AS). An explicit CONNECT SQL statement is no longer required in the application, because this is all done implicitly by DB2 for i when the three-part name is encountered.

A three-part name is a sequence of three SQL identifiers that specify the RDB name, the schema/library name, and the name of the object for use in an SQL statement.

SQL naming: <database-name>.<schema-name>.<object-name>

System naming: <database-name>/<schema-name>/<object-name>

In DRDA terminology, an AR is the code that handles the application end of a distributed connection. The AR is the application that is requesting data. An AS is the code that handles the
database end of the connection. For DRDA interoperability, shown in Figure 1, the AS can be a different DB2 for i database or it can be any database that supports the DRDA architecture. The DB2 for i database can also be an independent auxiliary storage pool (IASP) on the local system or a remote system.

Figure 1. DRDA interoperability inside and outside the DB2 family

A simple distributed relational database is shown in Figure 2 where the application runs on the application requestor, and the database being connected with is located on the application server. In the simplest of distributed relational database environments, a three-part name is run in an SQL statement on the AR, and an implicit DRDA connection is made to the AS where that SQL statement is run. Results are returned back to the AR from the AS over that DRDA connection. With three-part naming, more complex distributed relational database environments are possible and are discussed later in this article.

Figure 2. A simple distributed relational database connection
The examples in this article use the DB2 sample corporate database, created in schema CORPDATA, using the following SQL call:

```sql
CALL QSYS.CREATE_SQL_SAMPLE ('CORPDATA')
```

Now we will go through five ways to use DRDA to access remote data with increasing sophistication and flexibility.

**Option 1: Explicit connect**

Prior to three-part naming support, the user had to explicitly establish and manage the remote DRDA connection. The CONNECT SQL statement would be run specifying the target RDB. Depending upon how the remote RDB was configured, the user ID and password were passed in the statement or through a server authentication entry.

To connect over DRDA, an RDB directory entry needs to exist. The Add RDB Directory Entry (ADDRDBDIRE) command is run in the AR, creating the directory entry for the target AS. In this example, the *LOCAL RDB name, as defined at the AS, is specified in the RDB keyword as shown. The hostname or IP address is specified in the Remote Location (RMTLOCNAME) keyword. In this example, both have the same name, but this is not required.

```sql
ADDRDBDIRE RDB(LP24UT27) RMTLOCNAME(LP24UT27 *IP)
```

This following statement establishes an explicit connection to the relational database, LP24UT27.

```sql
CONNECT TO LP24UT27 USER ABC USING 'XZY'
```

To resume work with the local database after connecting to a remote RDB requires explicit re-establishment of that connection using the SET CONNECTION or CONNECT RESET SQL statement. These are laborious and tedious application tasks!

**Option 2: Three-part naming with actual RDB name**

Three-part naming gives users the ability to make the same connection above implicitly without using the CONNECT statement.

Previous limitations with using this option required specifying the actual RDB name directly as the first part of the three-part name. The actual RDB name is defined as the *LOCAL RDB name defined at the AS that you're intending to connect to over DRDA using SQL. The actual RDB name is one that does not specify an RDB alias.

Connecting through three-part naming also requires the existence of an RDB directory entry. The Add RDB Directory Entry (ADDRDBDIRE) command is run on the AR, creating the directory entry for the target AS. In this example, the *LOCAL RDB name, as defined at the AS, is specified in the RDB keyword as shown. The hostname or IP address is specified in the Remote Location (RMTLOCNAME) keyword. In this example, both have the same name, but this is not required.
If the database that is being connected to is LP24UT27, the SQL statement would appear as shown in the following example. The first part of the three-part name specifies the actual RDB name.

```sql
UPDATE LP24UT27.CORPDATA.EMPLOYEE
    SET SALARY = '60,000.00' WHERE EMPNO='000010'
```

A skilled programmer can immediately see the drawback of explicitly hard coding actual RDB names in your programs. RDB names are subject to change, and any change would require these SQL applications that use this RDB to be recompiled. Also, not all networks, where you want to use the program, have the same naming schemes for RDBs. Where is the application portability when using the actual RDB name hardcoded?

Unless you code to pass the correct RDB in, there is no application portability!

**Option 3: Using an SQL alias**

Adding a layer of object-level database abstraction is necessary to make your SQL applications more portable without having to pass in an RDB to your application. This abstraction also adds the benefit of hiding the actual database name where the sensitive data resides. Portability allows programmers to implement DRDA connectivity within their SQL applications without having to recompile when that database name changes or the program moves.

Users can create an "SQL alias" (not to be confused with "RDB alias") with the CREATE ALIAS SQL statement to get this added layer of object-level database abstraction and portability, but this method has its disadvantages as well.

Given the RDB directory entry created above in our example, a user would have to use the CREATE ALIAS SQL statement to gain a level of abstraction and portability. Any SQL aliases created using the CREATE ALIAS SQL statement need to be maintained.

This example allows for abstraction, but not portability.

```sql
CREATE ALIAS MYCORP.EMPLOYEE FOR
    LP24UT27.CORPDATA.EMPLOYEE
UPDATE MYCORP.EMPLOYEE
    SET SALARY = '60,000.00' WHERE EMPNO='000010'
```

**Option 4: Using an RDB alias in an SQL alias**

To add an additional layer of portability, an RDB alias can be specified in the CREATE ALIAS SQL statement. An RDB alias is the alternate name that is used to connect to an RDB. After an RDB entry has an RDB alias defined, users must reference the RDB alias name when connecting to that RDB. The exception to this rule is when a different RDB entry references the same target database and that RDB entry does not utilize an RDB alias.
Users specify the RDB alias using the second element of the RDB keyword for the ADDRDBDIRE or Change RDB Directory Entry (CHGRDBDIRE) command. The RDB directory entries can be viewed using the Display RDB Directory Entries (DSPRDBDIRE) command. The RDB directory entry list, accessed through the Work with Relational Database Directory Entries (WRKRDBDIRE) command, provides a central location for all DRDA RDB management on the AR.

In the following example, MYRDBALIAS is the RDB alias name for RDB LP24UT27.

```sql
ADDRDBDIRE RDB(LP24UT27 MYRDBALIAS) RMTLOCNAME(LP24UT27 *IP)
CREATEALIAS MYCORP.EMPLOYEE FOR MYRDBALIAS.CORPDATA.EMPLOYEE
UPDATE MYCORP.EMPLOYEE
   SET SALARY = '60,000.00' WHERE EMPNO='000010'
```

The biggest drawback of using the CREATE ALIAS SQL statement for this abstraction and portability is that under the covers, a distributed data management (DDM) file object is created to implement the connectivity. This implies that an additional object has to be maintained and backed up. You may ask yourself, why can't I avoid the negative aspects of the previous options and specify the RDB alias as the first part of a three-part statement directly?

The answer to the question is: *Now you can.*

**Option 5: Using an RDB alias directly**

With IBM i 7.2 and IBM i 7.1 running with DB2 PTF Group SF99701 Level 26 or higher, RDB aliases are supported within a three-part name. This enhancement allows for easier use of employing this additional layer of abstraction and portability. It also avoids the creation of additional DDM file objects (SQL aliases) on the AR that previously had to be managed.

Using the same example as before, the statements have now been simplified down as shown in the following example:

```sql
ADDRDBDIRE RDB(LP24UT27 MYRDBALIAS) RMTLOCNAME(LP24UT27 *IP)
UPDATE MYRDBALIAS.CORPDATA.EMPLOYEE
   SET SALARY = '60,000.00' WHERE EMPNO='000010'
```

When using the RDB alias directly in your SQL application, the only thing that has to be maintained is the RDB directory entry on the connecting database. If a database administrator wants to change the *LOCAL RDB name of the AS, developers no longer have to pass in the new RDB name or recompile all SQL applications on the AR to use the new RDB name. The administrator of the database has one place where they can simply change the RDB directory entry.

To connect to a different remote database using the same RDB alias name, simply change the RDB directory entry using the CHGRDBDIRE command. All subsequent use of the RDB alias name in the application will connect to the new remote database.
As you can see, the UPDATE SQL statement remains unchanged. In the following example, subsequent use of MYRDBALIAS RDB alias, after CHGRDBDIRE, will now connect to LP25UT27 instead of LP24UT27.

CHGRDBDIRE RDB(LP25UT27 MYRDBALIAS) RMTLOCNAME(LP25UT27 *IP)
UPDATE MYRDBALIAS.CORPDATA.EMPLOYEE
   SET SALARY = '60,000.00' WHERE EMPNO='000010'

Multitier DRDA connections

Another cool feature that RDB aliasing allows for with three-part naming is the ability to support multitier connections between an application requester and a server. In this more complex distributed database connection, the server that an application requester connects to is an application server, but any other server further downstream is called a database server (DS) as it does not interact directly with the application requester. In addition, to highlight its role as neither the database where a database request originates nor the database that performs the database function for the request, each application server or database server between an application requester and the final database server is also called an intermediate server.

A complex distributed relational database connection is shown in the following figure where the application runs on the application requestor, and the database management system (DBMS) running on the application server routes requests to a database server.

Figure 3. A complex distributed relational database connection

In this example, three-part naming is used on the AR to query results on the DS through one intermediate server. The application requestor connects to the application server, and the application server then connects to the distributed server where the three-part SQL statement is run. In this case, the AS acts as more of an intermediary which routes the SQL statement to the DS to be run. If the EMPLOYEE table exists at the DS, the query results are returned back to the AR seamlessly.

Setting up a multitier DRDA connection

To get the first connection of a multitier DRDA connection set up, a user specifies the RDB alias that exists at the AS in the first element of the RDB keyword of the ADDRDBDIRE command on the AR. Any RDB alias name of your choice can be used for the second element. The second element is the RDB alias to the RDB as known on the AR, and this is what the AR uses to connect with.
To set up the second connection for a multitier DRDA connection, a user specifies the actual *LOCAL name of the DS in the first element of the RDB keyword of the ADDRDBDIRE command on the AS. The second element of the RDB keyword must match what was used in the first element of the RDB keyword when setting up the first connection on the AR. The second element is the RDB alias to the RDB as known on the AS.

*Source-side aliasing* is the method of using an RDB alias as the second element of the RDB keyword. It is called source-side aliasing, because the RDB alias is used on the source side of the DRDA connection. Using an RDB alias in the first element of the RDB keyword is called *target-side aliasing*, because the RDB alias is used on the target side of the DRDA connection.

**Example:**

To fully enable the multitier DRDA connection support, the following prerequisite PTF's are required.

- SI54593 (7.1)
- SI54546 (7.2)

Step 1 is to add a relational database entry on the AR (SYSTEM1) which specifies an RDB alias (MYRDBALIAS) and RDB name (MYRDBALIAS2) that we will connect to on the AS (SYSTEM2). Step 2 is to add a relational database entry in the AS which specifies an RDB alias (MYRDBALIAS2) and RDB name (SYSTEM3) that we will connect to on the DS (SYSTEM3). Setting up the RDB entries in this manner allows connection hopping from the AR to the final DS using three-part naming.

**AR (SYSTEM1) -> AS (SYSTEM2) -> DS (SYSTEM3)**

**Step 1:** On the AR, add an RDB alias of MYRDBALIAS that references a remote RDB alias.

```sql
ADDRDBDIRE RDB(MYRDBALIAS2 MYRDBALIAS) RMTLOCNAME(SYSTEM2 *IP)
```

**Step 2:** On the AS, add an RDB alias of MYRDBALIAS2 that references a remote actual RDB name.

```sql
ADDRDBDIRE RDB(SYSTEM3 MYRDBALIAS2) RMTLOCNAME(SYSTEM3 *IP)
```

**Step 3:** On the AR, use the RDB alias MYRDBALIAS in your three-part statement.

```sql
UPDATE MYRDBALIAS.CORPDATA.EMPLOYEE
SET SALARY = '60,000.00' WHERE EMPNO='000010'
```

**Related topics**

As with all DRDA connections over TCP/IP, server authentication entries are necessary when explicit passing of the user ID and password are not possible. Server authentication entries allow you to link an RDB entry to a user ID and password used to connect with on a per user profile or group profile basis. Three-part naming requires server authentication entries to connect to remote
servers that require a user ID and password to authenticate. To specify a server authentication entry, users can use the Add Server Authentication Entry (ADDSVRAUTE) command. This is further described in the *Distributed database programming* topic available in the IBM i Knowledge Center.

When an application uses three-part name aliases for remote objects and DRDA access, a package for the application program must exist at each location that is specified in the three-part name. A package can be explicitly created using the Create SQL Package (CRTSQLPKG) command. If the three-part name alias is referenced and a package does not exist, the database manager attempts to implicitly create the package.

**Conclusion**

Whether you’re new to SQL or an experienced SQL programmer, I hope this article has given you insight into the options available to use DRDA more proficiently through three-part naming and RDB aliases. Understanding these concepts can enable you to improve your SQL database interoperability in your SQL applications or daily use, and also gain a level of database abstraction and portability.

**Resources**

- For more information about using special server authentication entries, read [Add QDDMDRDASERVER server authentication entry special value](#).
- To learn how to use group profiles with server authentication entries for DRDA connectivity, read [Simplified DDM and DRDA authentication entry management using group profiles](#).
- See our IBM i 7.2 Knowledge Center topic [Distributed database programming](#) for a more in depth look at using DRDA connectivity on IBM i.
- The [DB2 for i forum](#) provides a community discussion where you can ask questions and share solutions with your fellow developers and DBAs.
- Refer to the [Improve Your Data Center with Three-part Name Aliases](#) article for more information about using SQL aliases.
About the author

Nathan Skalsky

Nathan is a software engineer at IBM who specializes in the development of DRDA and SQL functionality in DB2 for i. When he's not at work, Nathan enjoys reading, traveling and exploring the outdoors with his family.