Achieve Improved Database Interoperability with SQL and RDB Aliases

by Nathan R Skalsky

In today's complex data-centric environments, where database connectivity is spread across numerous homogeneous & 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 easier manage remote DRDA connections within SQL applications and data centers.

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

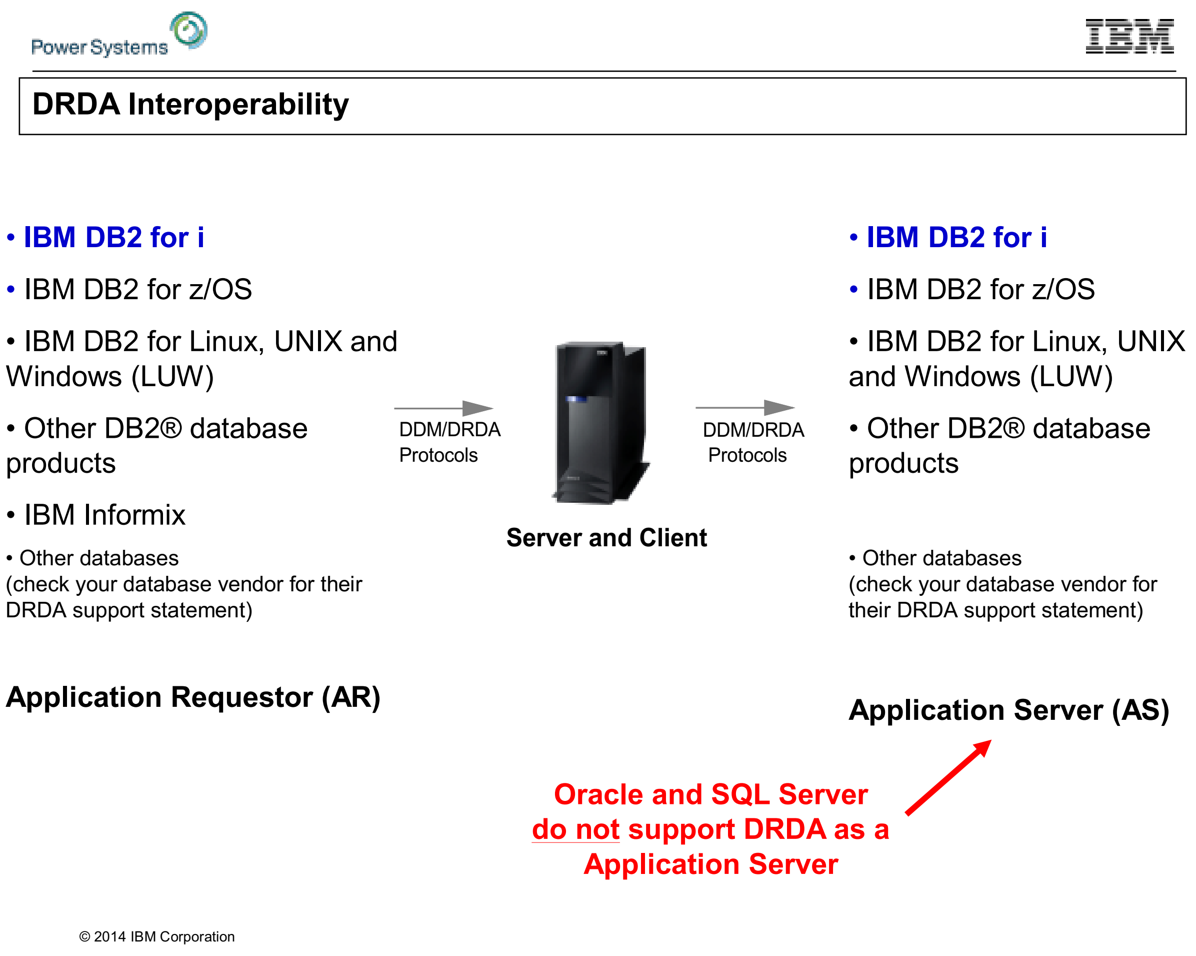
Three-part naming is the action of using a three-part name wherever allowed within a SQL statement. When a three-part name is used within an SQL statement, the initiator of the SQL statement is known as the application requestor (AR). The AR references distributed data over an implicit DRDA connection made to an application server (AS). An explicit CONNECT SQL statement is no longer required in the application, since 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 relational database name (RDB), 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. 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 ASP (IASP) on the local system or a remote system.

Figure 1: DRDA Interoperability inside and outside the DB2 family.



Power Systems

IBM

DRDA Interoperability

- **IBM DB2 for i**
- IBM DB2 for z/OS
- IBM DB2 for Linux, UNIX and Windows (LUW)
- Other DB2® database products
- IBM Informix
- Other databases (check your database vendor for their DRDA support statement)

Application Requestor (AR)

DDM/DRDA Protocols

Server and Client

DDM/DRDA Protocols

- **IBM DB2 for i**
- IBM DB2 for z/OS
- IBM DB2 for Linux, UNIX and Windows (LUW)
- Other DB2® database products
- Other databases (check your database vendor for their DRDA support statement)

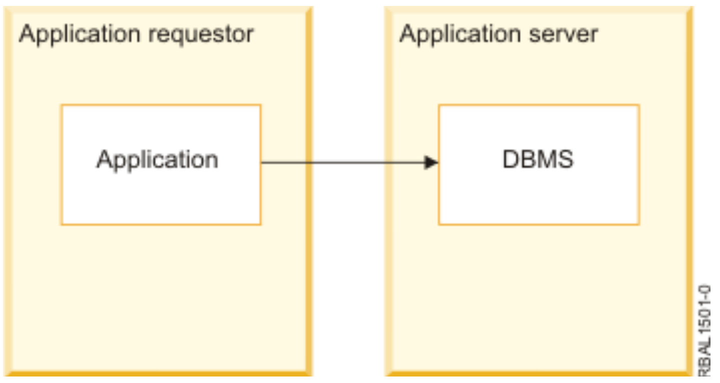
Application Server (AS)

Oracle and SQL Server do not support DRDA as a Application Server

© 2014 IBM Corporation

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 executed in an SQL statement on the AR, and an implicit DRDA connection is made to the AS where that SQL statement is ran. 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 will be 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, via the following SQL call:

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

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 executed specifying the target RDB. Depending upon how the remote RDB was configured, the userid and/or password were passed on the statement or via a server authentication entry. This statement below establishes an explicit connection to the relational database LP24UT27.

```
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 via 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 three-part naming required specify 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 your intending to connect to over DRDA via SQL. The actual RDB name is one that does not specify an RDB alias.

Connecting via three-part naming requires the existence of a RDB directory entry. The Add RDB Directory Entry (ADDRDBDIRE) command is executed 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 on the RDB keyword as shown. The hostname or IP address is specified on the Remote location (RMTLOCNAME) keyword. In this example, both have the same name, but this is not required.

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

If the database that is being connected to is LP24UT27, the SQL statement would appear as shown. The first part of the three-part statement specifying the actual RDB name.

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

A skilled programmer can immediately see the drawback of explicitly hard coding the 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 RDB’s. Where is the application portability when using the actual RDB name hardcoded?

Unless you add additional coding 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. The CREATE ALIAS SQL statement needs to be maintained and the application that codes it recompiled each time the RDB name changes.

This example allows for abstraction, but not portability.

```
CREATE ALIAS CORPDATA.EMPLOYEE FOR
  LP24UT27.CORPDATA.EMPLOYEE

SELECT * FROM CORPDATA.EMPLOYEE
```

Option 4: Use an RDB alias in an SQL alias

To add an additional layer of portability, an RDB alias can be specified on the CREATE ALIAS SQL statement. An RDB alias is the alternate name that is used to connect to an RDB. Once 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 that RDB entry does not utilize an RDB alias.

Users specify the RDB alias via the second element of the RDB keyword for the ADDRDBDIRE or Change RDB Directory Entry (CHGRDBDIRE) command. The RDB directory entries can be viewed via the Display RDB Directory Entries (DSPRDBDIRE) command. The RDB directory entry list, accessed via Work with Relational Database Directory Entries (WRKRDBDIRE) command, provides a central location for all DRDA RDB management on the AR.

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

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

CREATE ALIAS CORPDATA.EMPLOYEE FOR
  MYRDBALIAS.CORPDATA.EMPLOYEE

SELECT * FROM CORPDATA.EMPLOYEE
```

The biggest drawback of using the CREATE ALIAS SQL statement for this abstraction and portability is that under the covers, a 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 all of this overhead 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 backed up and maintained.

Given the previous examples, the statements have now been simplified down to the following:

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

SELECT * FROM MYRDBALIAS.CORPDATA.EMPLOYEE
```

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, they 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 via 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 SELECT statement remains unchanged. In our example below, subsequent use of MYRDBALIAS RDB alias, after the CHGRDBDIRE, will now connect to LP25UT27 instead of LP24UT27.

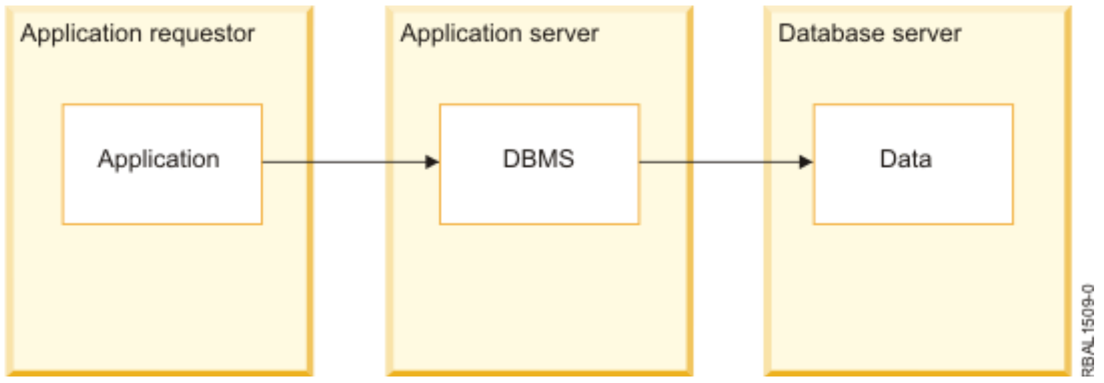
```
CHGRDBDIRE RDB(LP25UT27 MYRDBALIAS) RMTLOCNAME(LP25UT27 *IP)

SELECT * FROM MYRDBALIAS.CORPDATA.EMPLOYEE
```

Multi-tier DRDA connections

Another cool feature that RDB aliasing allows for with three-part naming is the ability to support multi-tier 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 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 executed. In this case, the AS acts more of an intermediary which routes the SQL statement to the DS to be executed. If the EMPLOYEE table exists at the DS, the query results are returned back to the AR seamlessly.

Setting up a multi-tier DRDA connection

To get the first connection of a multi-tier 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 desired RDB alias name can be used for the second element. The second element is the RDB alias to the RDB as known on the AR – This is what the AR uses to connect with.

To set up the second connection for a multi-tier 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." Conversely, it is called target-side aliasing, because the RDB alias is used on the target-side of the DRDA connection.

Example:

Prerequisite PTF's:
  SI54593 (7.1)
  SI54546 (7.2)

Step 1 is to add a relational database entry on the AR (SYSTEM1) which specifies a 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 on the AS which specifies a 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 via 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.

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

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

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

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

```
SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,
 CASE
  WHEN EDLEVEL < 15 THEN 'SECONDARY'
   WHEN EDLEVEL < 19 THEN 'COLLEGE'
   ELSE 'POST GRADUATE'
 END
FROM MYRDBALIAS.CORPDATA.EMPLOYEE
```

Related topics:

As with all DRDA connections over TCP/IP, server authentication entries are necessary when explicit passing of the userid and password are not possible. Server authentication entries allow you to link an RDB entry to a userid 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 userid 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 will attempt to implicitly create the package.

Other enhancements not discussed in this article that can be used in conjunction with three-part naming are:

Add QDDMDRDASERVER server authentication entry special value

Simplified DDM and DRDA authentication entry management using group profiles

CREATE TABLE with remote SUBSELECT

INSERT with remote SUBSELECT

Improve Your Data Center with Three-part Name Aliases