

Database Interoperability

ELIMINATING DBMS BARRIERS

**W H I T E
P A P E R**

February 2000

**PRINCETON
SOFTECH**

Contents

1. Database Interoperability: The Challenges	1
2. Introducing Move for Servers	4
3. Interoperability in Action	6
4. Data Types and SQL Support	12
5. Transforming Application Data to eData	15

1. Database Interoperability: The Challenges

Remember back when relational database systems promised to make your life easier? Several aspects of the relational model, it was argued, would combine to produce a new era of nimble, manageable databases.

First, data would be stored in tables, a universally understood method of organizing information. Next, there would be no “pointers” or navigational information embedded in the database — nothing but pure, unadulterated data. (Of course, the data would have to be normalized in order to make it clear and useful.) Changes that typically disrupted older DBMSs, such as adding new data fields, new relationships and new record or segment types would be accommodated transparently in a relational DBMS. This would leave existing programs unaffected and they would continue to operate properly.

Beyond this, relational systems would include a standardized language for defining databases and manipulating the data they contain. Derived from mathematical theory, this language would introduce an element of academic precision to the world of data manipulation, while at the same time be relatively easy to learn and use. In theory, data could be moved from one DBMS to another and programs accessing that data would still work correctly.

That sums up the initial promise of relational DBMS. Since then, thousands of applications have been implemented using the relational data model. What have we got to show for the effort? Has the promise of relational technology been fulfilled?

On balance, one would have to give relational DBMSs pretty high marks. Compared to the previous generation of database technology, relational systems are indeed much easier with which to work. Organizations have been able to grow and extend their DBMS-based applications without the extensive rewriting and reworking activities that were commonplace in the past. End users have gotten better access to the decision-making data they need. Though older DBMS technologies are still in use (precisely because it is prohibitively expensive to replace them), virtually every important new application and package of the past decade has been based on the relational data model.

So the database wars are over... and the relational model has won.

But large-scale deployments of relational technology are not without their problems. The relational database revolution occurred at the same time as the movement toward client/server applications. Many companies implemented mission-critical applications using relational DBMSs, but they also saw a proliferation of platforms with applications using other relational DBMSs.

The result is that most large corporations today are actively using multiple relational DBMSs. Corporations now need to support eBusiness programs, which means that enterprise data must be transformed to eData — and this often means moving data from one relational DBMS to another.

For example, DB2 may support a mission-critical production system, while Oracle is used for the ERP system and other client/server applications. SQL Server may house a database used for web access. The web database may contain catalog and inventory data that is derived from the DB2 and Oracle systems; a corporate data warehouse may hold information that originated in both DB2 and SQL Server; updates to the web database must be propagated back to production databases. To accomplish all of this, the IT organization needs a data movement strategy that is accurate and scalable.

Further, accurate relational data migration is an important component of application testing. In an eBusiness world, you need failsafe testing as customers are more demanding than ever before. At the same time, you need speed: you can't afford to duplicate your entire corporate database to test a new eBusiness application or spend time writing extract programs.

But despite the conceptual simplicity of the relational data model, it is not easy to move data from one relational DBMS to another. Here are the key challenges that must be overcome:

- Complex Data Models

Moving the right data is the first priority in a data movement strategy. But production databases have dozens, hundreds, and in some cases, thousands of tables. Relationships connecting these tables are many and often they are managed from within the application.

Finding a way to create “referentially intact” subsets of these databases — along with senior SQL programmers who can write the necessary extract programs — is a major issue for many organizations. As a result, most companies find it very difficult to accurately move data between applications and platforms. Data quality suffers. Corruption can occur as a byproduct of just trying to move the data.

- SQL Dialect Differences

Although SQL has done much to standardize access to relational databases, the different DBMS vendors have not implemented SQL in the same way. Inconsistencies can arise when data is moved between DBMSs or the same SQL statements are applied to two different DBMSs.

- Data Type Differences

Every relational DBMS supports a range of data types; columns can be numeric, character, date and so forth. However, each vendor has implemented data types differently. Numeric precision and rounding may differ. Sometimes one DBMS has no equivalent to a special data type supported by another DBMS. This presents a major challenge when data is moved between systems, because data can potentially be lost or eroded.

The foregoing demonstrates why the next advancement in relational technology is to meet the challenge of *interoperability*. This challenge is being met today by one software product: Princeton Softech's Move for Servers™.

2. Introducing Move for Servers

The interoperability features of Move for Servers makes it possible to easily transport data between heterogeneous DBMSs.

Getting the Right Subset of Data

As mentioned above, enterprise-scale relational databases often contain hundreds of interrelated tables. The relationships connecting these tables are often very complex. Accurately following the chain of relationships across dozens of tables to acquire the right subset of data — and then maintaining that data in a referentially intact state — is difficult even for the strongest SQL programmer.

Move for Servers is based upon Princeton Softech's Relationship Engine™ technology, which has the ability to create subsets from even the most complex database — no matter how many tables or relationships are involved. Move for Servers always gets 100% of the right data, referentially intact and complete, every time.

Move for Servers also maintains a metadata directory, which is used to store knowledge of the data model and its extended relationships — relationships that are usually not documented in the DBMS catalog, data dictionary or system tables, but are known to the applications. In this way, users are able to capitalize on the referential intelligence available to Move for Servers, without having to research a very complex data model every time there is a new data movement need.

Creating the Target Data Model

When moving data from one DBMS to another, if necessary Move for Servers can automatically define the tables, data elements and relationships to the target DBMS. Thus, Move for Servers preserves both the data and the data model, completely intact.

Dealing with Data Type Inconsistencies

Move for Servers can translate data seamlessly between any of its supported DBMSs. Its approach is to handle most tasks (e.g., EBCDIC to ASCII conversion) automatically, while stopping to notify the user and request guidance when the intent is unclear or loss of data (e.g., significant digits would be truncated) may result.

Handling Variations in SQL

Largely because of differences in data types, the SQL DDL statements used to create the source database may imply different things to the target database. Move for Servers provides annotation to inform the user of these variations. In many cases, the appropriate action is obvious, which Move for Servers will suggest by default; in other cases, the user will need to tell Move for Servers what actions to take.

3. Interoperability in Action

In terms of DBMS support, Move for Servers can accomplish three kinds of data movement:

1. Extract from DBMS A and Insert to DBMS A (homogenous data movement)
2. Extract from DBMS A and Insert to DBMS B (heterogeneous data movement)
3. Extract from DBMS A and DBMS B and Insert to DBMS C (federated data movement)

Interoperability is a fully integrated part of Move for Servers. In essence, the same procedures and functions that are used in homogeneous data movement are also used when heterogeneous DBMSs are involved. The first step in moving data is to create an Access Definition:



Figure 1: Access Definitions Control the Data Movement Process

The Access Definition identifies the subset of data that is being carved out of the source database. It contains the rules by which Move for Servers will traverse the source database to assemble a referentially intact data subset. There is no limit to the number of tables or relationships that may be included in a single Access Definition.

Often, Move for Servers is used to accomplish homogeneous data movement:

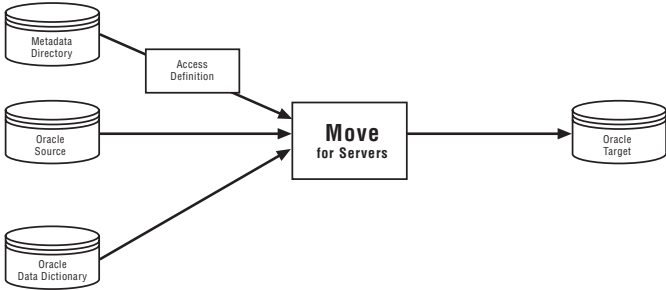


Figure 2: Homogeneous Data Movement

Move for Servers always checks the DBMS metadata to ensure that no changes have been made to the data model since the time the Access Definition was created. It can dynamically incorporate changes if necessary. Figure 2 depicts the movement of data between two Oracle databases.

In *heterogeneous* data movement, the source and target DBMSs are different. The source DBMS and destination DBMS can be any of those supported by Move for Servers (presently they include Oracle, Sybase, DB2 UDB, DB2 UDB for OS/390 and SQL Server). Users can move data from Oracle to Sybase, from UDB to Oracle or any combination of the supported DBMSs. Source and destination DBMSs are specified on the Move for Servers “Table Map,” shown below in Figure 3 which depicts movement between a UDB and an Oracle database. While the example shows that the two databases have the same table names, this is not required by Move for Servers.

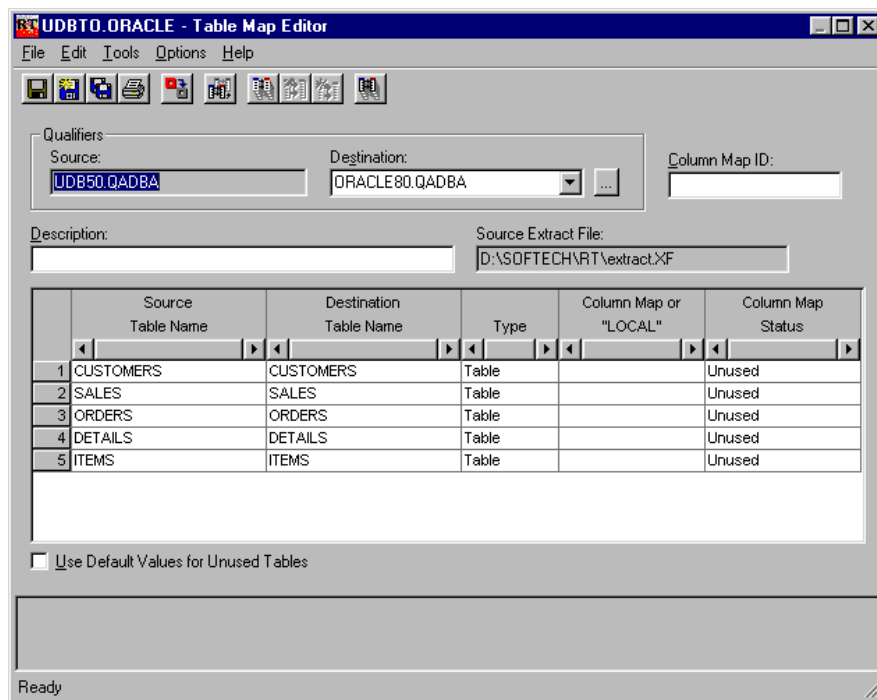


Figure 3: Moving Data From UDB Into Oracle

When Move for Servers detects the fact that heterogeneous data movement is being requested, it automatically invokes special functions to ensure that data is moved safely and accurately while differences between the two DBMSs are taken into account.

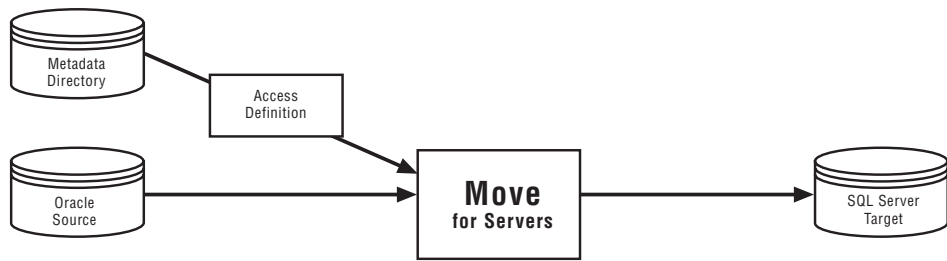


Figure 4: Heterogeneous Data Movement

Operationally, there is no difference between homogeneous and heterogeneous data movement: the Access Definition dictates how the precise subset of data is to be extracted from the source DBMS and the Table Map dictates what type of tables are used as targets. Just as with homogeneous data movement, Move for Servers can either create new tables and relationships, refresh existing tables or update existing tables at the target.

Another kind of data movement supported by Move for Servers is “federated” data movement. In this case, data from more than one source DBMS is combined and then moved into a target DBMS. For example, if customer data is maintained in a DB2 database but certain history is kept in an Oracle data warehouse — and the customer number can be used to match the customers to their history — then Move for Servers can join that data and transport it to a target database, which can be managed by a third DBMS such as SQL Server (Figure 5).

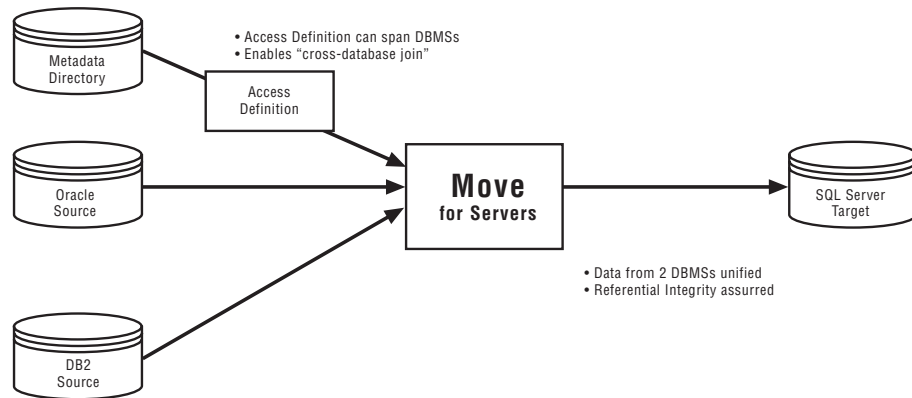


Figure 5: “Federated” Data Movement

Once again, the user continues to work with Move for Servers’ familiar Access Definition, Table Map and associated facilities. Move for Servers detects that federated data movement with heterogeneous DBMSs is requested and automatically takes care of the details that are specific to each individual DBMS. Clearly in this case, the relationships used for the federated extract would have to be supplied to Move for Servers since they will not be found in the database’s metadata.

The user can see that different tables are being joined from heterogeneous sources by noting the identity of the DBMSs in the Access Definition. Figure 6 below shows an example.

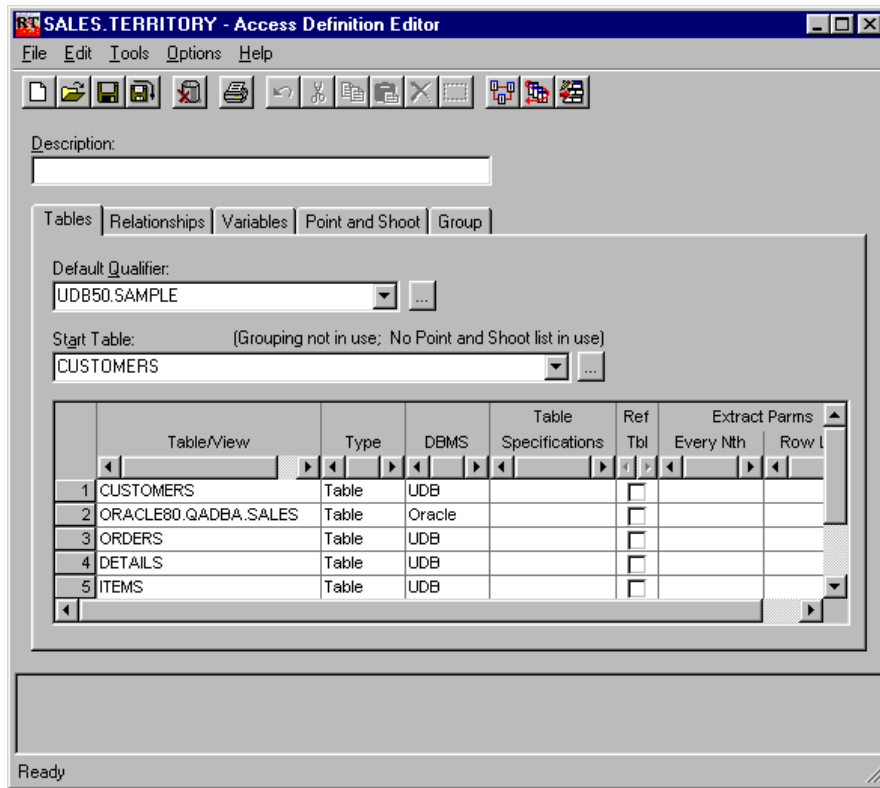


Figure 6: An Extract That Combines Customer Data From UDB With Sales Data From Oracle

4. Data Types and SQL Support

When data is moved from one DBMS to another, certain anomalies can occur. Despite the simplicity of the relational data model and its relative consistency from DBMS to DBMS, on the level of nitty-gritty details each vendor has made its own implementations. The result is that one cannot simply move data between DBMSs and expect all data to reach the destination safely.

For every column, an assessment of the possible consequences of moving to a new DBMS must be made. The specific issues are many and include things like these:

- Oracle does not allow the user to distinguish between a NULL value and a zero length character string. Since many other DBMSs support this distinction, the ambiguity must be resolved when moving data into Oracle.
- DB2 supports fractions of a second to 6 decimal places while Sybase tracks fractional seconds to 1/300. Users need to review these different levels of precision to avoid potential rounding errors.
- Oracle does not support NULL values in character columns. Users need to be aware of this and review data originating from DB2 and Sybase for potential invalid data transformation.
- Sybase has introduced additional data types including money, small money and bit. Users need to translate this information into correct formats.

Move for Servers automatically assesses the impact of heterogeneous data movement and presents the user with useful information and options:

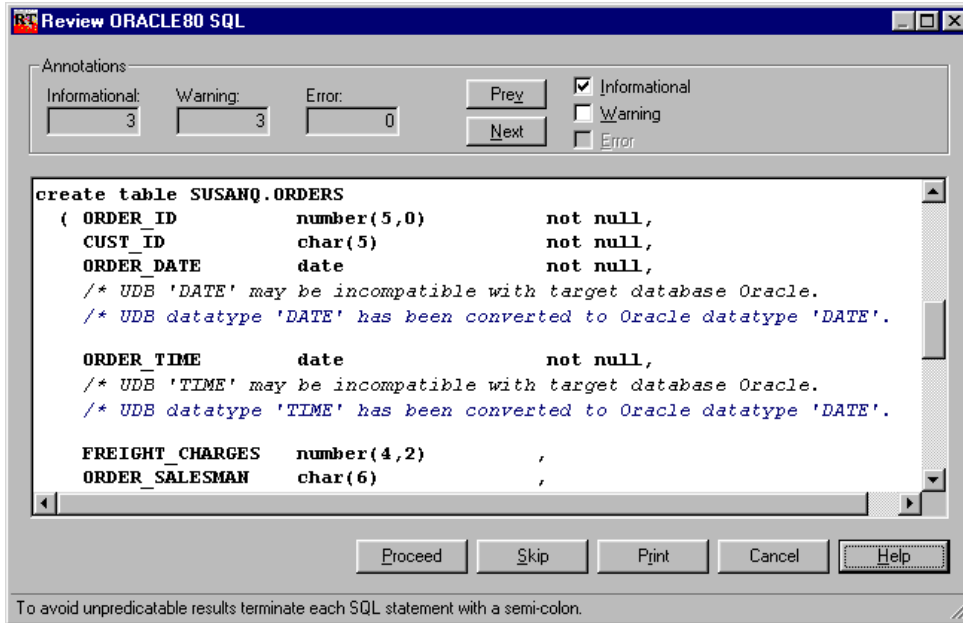


Figure 7: Sample Report Showing Warning Messages in Heterogeneous Data Movement

For example, Figure 7 shows part of a report that results from a heterogeneous data movement operation. In this case, data was moved from UDB into Oracle. Because the Oracle tables did not already exist at the destination, Move for Servers automatically defines them using standard Oracle SQL DDL statements.

In Figure 7, because there are differences between the way UDB and Oracle handle date and time fields, the user is alerted. (Since UDB supports dates from 1/1/01 AD through 12/31/9999 AD, and Oracle's date field allows dates from 1/1/4712 BC to 1/1/4712 AD, there is the potential for some valid dates in UDB to be invalid in Oracle. This is an example of a case where there would rarely be an actual problem, but the potential for one theoretically exists.) Move for Servers has three levels of messages that it prints as comments in the SQL DDL created during heterogeneous data movement operations:

- Informational

Move for Servers made some necessary and obvious change as it prepared the DDL for the converted data. These changes should not have any impact on the user, but the messages are issued so that a complete and clear audit trail is preserved.

- Warning

Due to data type differences, Move for Servers was forced to make a change that could impact the user. Often this impact is more academic than real, such as the conversion of date column shown in Figure 7 — as long as all the dates fit within the range supported by both DBMSs (which is the overwhelming likelihood), then the user will see no difference. However, it is important to alert the user to the possible erosion of data — which is what this message does. Analysis of the source data can always be done to ensure that data erosion is avoided.

- Intervention required

In this case, Move for Servers has encountered a serious and unavoidable incompatibility between the two DBMSs. Data movement can still be done, but the user must dictate what action Move for Servers must take — such as to drop the incompatible column or to convert its data into a character string in the target database.

Users can confidently employ Move for Servers for heterogeneous data movement tasks. Move for Servers always validates the potential impact on each column, taking into account the source DBMS, target DBMS and the data type of the column. Users are presented with a full audit trail and can specify how Move for Servers should process each column.

5. Transforming Application Data to eData

Heterogeneous platforms, heterogeneous databases and distributed computing are an increasing way of life for enterprise IT organizations. More and more, data must be moved from one platform to another and from one DBMS to another. When moving data, users want speed, accuracy, scalability and safety.

Moving critical information to where it is most needed can help sell products, deliver better customer service or cut supply chain costs. It's what we call transforming application data to eData — putting data at the maximum point of leverage for the organization.

Move for Servers delivers these benefits and enables IT organizations to save money while making it easy to bridge the gaps between diverse data platforms.

princetonsofttech.com

111 Campus Drive
Princeton, NJ 08540-6400
Toll free **800.457.7060**
Phone 609.688.5000
Fax 609.497.0302

**PRINCETON
SOFTECH**