ORAtips Journal January 2006 Volume II Issue 1

Migrating from SQL Server 2000 to Oracle® 9i

By Blake Couch

Editor's Note: SQL Server has plenty of uses as a platform for applications, from Microsoft and SAP to People Soft. Given the confusion surrounding Oracle Fusion, and the constant rumors about the future direction for PeopleSoft and JD Edwards applications, it seemed timely to present an article discussing the migration from SQL Server to Oracle. First time ORAtips contributor and Oracle veteran Blake Couch has "been there", "done that", and provides his unique insight on this process, offering tips for both the rookie and the seasoned DBA professional alike.

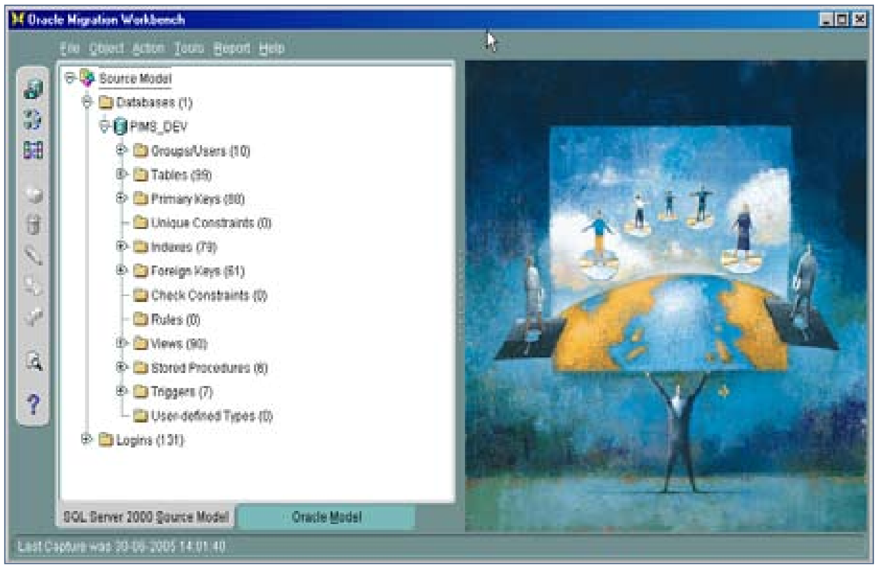

Figure 1: Oracle Migration Workbench

Introduction

In the business world, a relational database management system is often virtually invisible, understood by and known to only the database practitioners who maintain it. Yet it is frequently the foundation of the business processes upon which an organization depends. Which RDBMS to use may seem a thoroughly arcane decision to all but those same practitioners, but that choice can have far-reaching implications, even for the end user. There are perhaps as many reasons for switching from one RDBMS to another, as there are organizations that use them, but the differences between systems are substantial enough to make switching a process that merits careful preparation.

This article presents information relevant to the differences between SQL Server 2000 and Oracle 9i, and will show how to handle those differences when converting from SQL Server to Oracle. What I present here is not necessarily true for other, especially later, versions, nor is it meant to be exhaustive of the differences between the two systems.

The Oracle Migration Workbench

There's an awful lot of grunt work to be done in migrating an entire schema along with its data from one RDBMS to another. Oracle supplies a tool called the Oracle Migration Workbench (OMW), available at **http://www.oracle.com/technology/tech/migration/workbench/index.html**, which takes away a lot of the drudgery of scripting DDL for table, index, constraint, view, and user definitions.

At a high level, it converts stored procedures and triggers – not perfectly, but it does give you a starting point. It supports a wide variety of source database systems, including SQL Server, Sybase, Informix, DB2, and MySQL. The OMW targets both Oracle 9i and 10g. It allows you to tweak the logical Oracle model it creates for you before it generates the scripts you will use to create your physical model. It generates data extract scripts, using BCP for SQL Server, creates your SQL*Loader script and control files, and yes, it is a free download.

Based on my experiences, I highly recommend you download the OMW and take full advantage of it. The OMW will save you an enormous amount of time and effort, but it is only the beginning of your migration journey. You've still got a lot of work to do, so please keep reading.

Case Sensitivity

The differences between SQL Server and Oracle are quite numerous and could, indeed, fill an entire book-length volume. Which of those differences are most pertinent to the task of converting from one to the other? The single largest issue in this realm is case sensitivity. SQL Server is by default not case-sensitive when it comes to data values. Oracle is case-sensitive. This has ramifications, which extend from the database right on up to end-user applications.

In SQL Server, all of the following are considered equivalent data values:

primary key, Primary Key, PriMary keY, PRIMARY KEY

If one does a select on a column in a SQL Server table where any of those values is specified in the WHERE clause, all rows containing any one of them will be returned in the result set. Likewise for joins: a join on tables based on columns containing a mix of those values will result in matches on all of them.

In Oracle those four values are separate and distinct. Executing the following statement in Oracle

```
SELECT * from myTable where myKey =
'primary key'
```

will not return rows where myKey contains Primary Key, PriMary keY, or PRIMARY KEY. Now if you never allow your users to store data values in heterogeneous or mixed case, case sensitivity will not be an issue for you in a conversion, but in practice, where heterogeneous case is allowed by the system, it is likely to be found in the data.

How does one handle this situation in converted data? You could search through all your key values for case variations and handle them individually, but, as a practical matter, this could be extremely time-consuming. The most expedient solution I have found is to update all non-numeric keys, both primary and foreign, to either all upper or all lower case. Which you choose is purely personal preference, as long as you're consistent with it.

Generating a script to uppercase all your primary keys is as simple as capturing the result set from

```
SELECT 'update ' || c.table_name || ' set
' || cc.column_name || ' = upper(' ||
cc.column_name || '); '
FROM user_constraints c, user_cons_columns cc
WHERE c.constraint_name = cc.constraint_name
AND c.constraint_type = 'P'
AND c.owner = 'SchemaOwner';
```

and running it as a script in SQL*Plus.

Now that your data is in good shape, its time to turn your attention to case issues in your applications. Finding and fixing these is likely to be a much larger task. You need to look for instances of hard-coded key values in SQL WHERE clauses and in comparisons in code between constants and values returned from the database. If, for example, you've chosen to uppercase all your keys, any code that relies on lower- or mixed-case values is going to break. I found it useful to generate a list of distinct key values and then grep all my source code for those values.

Capturing the output from

```
SELECT 'select distinct ' || column_name
|| ' from ' || table_name || '; '
FROM USER_CONS_COLUMNS
WHERE position IS NOT NULL
```

will give you a script you can run to generate a list of key column values. You can use this list to do a case-insensitive search through your source code using grep or a similar search tool.

Cascading Updates

SQL Server supports cascading updates. Oracle does not. This means that to the extent your SQL Server-based applications rely on the RDBMS to propagate changes to key values where foreign keys exist and referential integrity is enforced, you must find another way to support that functionality if you're going to migrate to Oracle. The OMW will not help you in this area of your migration process. Writing triggers to mimic the cascading update process is a bit tricky, but is the only practical way I have found to handle this problem.

The problem with writing triggers is, if referential integrity is enforced on the column in question, you cannot update the parent row without violating referential integrity in the associated child row. Similarly, you cannot update the child row before the parent row or the same violation will occur. If you could disable and then re-enable referential integrity from within a trigger, you could have your trigger update the child rows in between, but Oracle will not allow you to do that. DDL inside a trigger is a no-no. There are some clumsy and resource-consuming workarounds for this problem, involving temporary rows and/or tables, but I have found the best solution is simply to remove the column in question from both the primary key in the parent and the foreign key relationship in the child. Then you can write your trigger to update the child in the After Update event of the parent.

The argument in support of this approach, based on relational theory, is that primary key values are supposed to be immutable. Therefore, if you have a column in a multi-column primary key that is subject to change over time, it does not belong as part of the primary key. As a practical matter, you may find that your primary key is no longer unique once you remove that changeable column from it. If that is the case, you will need to substitute a sequence value or some other existing column that makes the key combination unique. At any rate, you should probably take a hard look at the design of the tables in question if, in fact, you do encounter this issue.

Maximum Identifier Length

In Oracle, all identifiers – table names, column names, constraint names, index names, etc. –are limited by the RDBMS to 30 characters. As unrealistic as it may seem, that is the Oracle world. All of your SQL Server identifiers must be converted to 30 characters or fewer, and what is worse, all references to any identifiers that are changed must be updated as well! This no doubt strikes you as a daunting task, especially if you happen to use a lot of long names, but the OMW will handle this for you. All of your schema items and references to them in your stored procedures will be updated by the OMW in your new Oracle schema. This leaves you with the task of fixing your application code.

To get a list of your SQL Server column names whose lengths are greater than 30 characters, run this script

```
SELECT distinct column_name
FROM information_schema.columns
WHERE len(column_name) > 30
```

against your database in Enterprise Manager or whatever you use to run ad hoc queries on your data. You use the list of columns generated to search your application source code for problem column references. You may want to check your other information schema views, like Key_Column_Usage, for other instances of identifiers that are too long for Oracle. Foreign key constraint names tend to be especially problematic, because people like to incorporate two table names and at least one column name in them.

Converting Date Columns

Oracle's default date format "DD-MON-YY" is a bit peculiar and not conducive to loading date values straight out of SQL Server. Fortunately the OMW handles this for you by creating views in your SQL Server instance that extract date column data into a user-friendly format. The SQL*Loader process, which gets your data over into Oracle, then resets Oracle's default date mask to this same format before it loads any data.

Leading/Trailing Spaces

SQL Server by default will ignore trailing and leading spaces in key values when performing lookups and joins. Need I mention that Oracle does not do this? Once again, if the RDBMS allows it, and you do not take explicit steps to prevent it at the application level, chances are you will wind up with what Oracle considers "dirty data" when your migration is done. In the Migration Steps section of this article, I show how to create a script that will trim leading and trailing spaces from all of your key values.

All of your SQL Server identifiers must be converted to 30 characters or fewer....

Timestamp Columns

SQL Server TIMESTAMP columns do not convert well at all to Oracle. The OMW will attempt to turn them into LONGs, but this really does not give you what you want. The problem is that there is no data type equivalent to the SQL Server TIMESTAMP in Oracle 9i. You are really on your own when it comes to converting these columns.

Stored Procedures

Converting T-SQL stored procedures to Oracle's PL/SQL is a difficult task. Both systems have their own proprietary extensions to SQL, and, of course, their own peculiarities of syntax. The OMW takes on this task and will do a creditable job for you, but you will want to test thoroughly any converted functions, procedures, and triggers. Chances are you will want to make changes to the generated code, for performance tuning if no other reason. OMW is particularly helpful, creating functional equivalents to SQL Server's proprietary SQL functions (e.g., string manipulation functions like MID and INSTR) in PL/SQL and then aliases them for you. This relieves you of the task of converting code and SQL statements that use those functions.

Boolean Data Types

There is no Boolean data type in Oracle (nor is there one in SQL Server, but the bit type is more or less functionally Boolean). All of your Booleans from SQL Server will have to be converted to a non-Boolean type in Oracle. Typically such data is converted to CHAR(1), with TRUE stored as a "Y" and FALSE stored as an "N", but I prefer the method employed by the OMW. It brings Booleans over from SQL Server as 1 for TRUE and 0 for FALSE and stores them in a NUMBER(1) column. Either way, you will still need to make changes to any applications that make Boolean comparisons on these columns.

For example, code like "IF myBoolean THEN..." or "IF myBoolean = TRUE THEN..." probably will not work once your back-end has been converted to Oracle (some programming languages will support "IF myBoolean", but many will not). You'll need to change your code to "IF myBoolean = 1" or "IF myBoolean = 0".

To locate your SQL Server bit columns, you can use the following script:

```
SELECT table_name,column_name
FROM information_schema.
columns
WHERE data_type='bit'
```

Migration Steps

Many of you Oracle folks reading this are also regular users of TOAD. Let me offer a word to the wise: avoid using TOAD to run the SQL scripts that I mention in the migration steps, especially if your data is millions of rows. Use SQL*Plus instead. In my experience, TOAD has a tendency to choke and hang on longer-running scripts. You will want to log all your SQL*Plus sessions, so remember to spool your output from those sessions. Setting timing ON can be useful as well.

The three basic steps in migrating from SQL Server to Oracle are:

1. Create the Oracle schema

2. Extract your data from SQL Server

3. Load the data into Oracle

Using the OMW gives you control over the process because of its granularity, and by handling a lot of the details for you. Below are the steps, in the order they should be executed, as they break out in an OMW migration:

Step 1: Capture the SQL Server schema with OMW and create the SQL Server data model

This step can be performed either online, using ODBC connectivity, or offline, using a .bat file. Either method will create several dozen XML and DAT files, which describe the SQL Server schema.

Step 2: Make any necessary changes to the SQL Server data model

This is your opportunity to tweak your schema before you migrate it to Oracle. For example, you may have some column or table names that were not particularly well chosen when you first defined the schema. You can fix them before they make their way into your new Oracle instance.

Step 3: Create the Oracle data model in the OMW, based on the SQL Server model created in Step 1

Both the SQL Server and the Oracle data model are stored in the OMW's internal metadata repository. You can also store this metadata in an actual Oracle database, if you so desire. After generating the Oracle model, you again have the opportunity to tweak it before generating the scripts to create the physical model. For instance, you might want to change the tablespace names to utilize tablespaces that already exist, or you might want to update the schema owner name.

Step 4: Generate the Oracle schema build and data migration scripts in OMW

During this step, a series of scripts will be created that use the SQL Server BCP utility to extract data into flat files. It will also create all your Oracle schema elements with DDL instructions and populate your new Oracle tables using SQL*Loader utility.

Step 5: Run the Oracle schema build scripts

It is advisable after this step to disable temporarily the primary key, foreign key, and not null constraints in your new Oracle database. Note that foreign key constraints must be disabled before primary key constraints or you run into dependency issues, and that not null constraints should be disabled last. While your data may be perfectly clean from the SQL Server point of view, it will likely be dirty to Oracle, for the reasons I've outlined already. If you leave constraints in place while you attempt to load data, you will find yourself constantly restarting the data migration in order to address constraint violations. I have found it is easier to get all the data loaded into Oracle first and then re-enable constraints while noting which ones fail to enable. This way you see where your data needs fixing and you can handle it from within the database where you will have the power of SQL and the database engine to help with the corrections.

To disable constraints, capture the output from

```
SELECT 'alter table ' || table_name || '
disable constraint ' || constraint_name
|| ';'
FROM user_constraints
WHERE owner = 'myOwner'
```

and run as a script in SQL*Plus. Remember to reorder the lines produced in order to get the foreign keys first.

Likewise you should disable any insert or update triggers in your new Oracle database before attempting to load data. Otherwise, you'll slow down the load process, and you could find yourself with data anomalies, especially if your triggers generate rows in other tables. What you are trying to do is create a mirror image of your SQL Server data in Oracle. You're not interested in duplicating the normal insert and update procedures during the data migration.

Of course, if you've got a powerful ETL tool at your disposal, and it is one you are already familiar with, you could be better off using it to migrate your data. Remember, though, that the OMW is free, and is easy to learn and use.

Step 6: Create the views in SQL Server that will be used to extract data formatted for Oracle using the Create Views script generated in Step 4

An obvious prerequisite – run this script as a user with create view rights. There is another script you'll run later that deletes all these views for you.

Step 7: Extract your SQL Server data using the script created in Step 4

Use the bcp_extract.bat script created for you by the OMW. It makes a call to the SQL Server bcp utility for every view you created in Step 6, and for every table in your database where a view is not needed, dumping the data for each into a separate flat file.

Remember.... Oracle Migration Workbench is free and easy to use!

Step 8: Load the extracted data into Oracle using the SQL*Loader script created in Step 4

Two things to consider during this step: use direct path loads for your larger tables, and use the parallel option if your server has multiple processors. Both options must be added manually to the generated SQL*Loader scripts.

Step 9: In a SQL*Plus session, run a script to change all your key values in Oracle to either upper or lower case

Capture the output from

```
SELECT 'update ' || table_name || ' set '
|| column_name || ' = upper(' || column_name || ');'
FROM USER_CONS_COLUMNS
WHERE position IS NOT NULL
```

This produces a script you can run to accomplish this step. It is something of a brute-force method, since it does not exclude numeric keys, but Oracle will simply ignore the update commands for those keys.

Step 10: Trim leading and trailing spaces from all your key columns in Oracle

You can use the SELECT from Step 9 to generate the update script for this step by changing "upper" to "trim".

Step 11: Re-enable constraints and triggers in Oracle

For the constraints, you can take your "disable constraints" script, globally replace the word "disable" with "enable", change the order of the lines so that primary key constraints are enabled first, and run the resulting script in SQL*Plus.

Step 12: Run the Drop_View.sql script to remove your temporary views from SQL Server

And that is all there is to it!

Conclusion

There's no getting around it: migrating from SQL Server to Oracle is complicated and exacting. Scrupulous attention to detail is required if you are going to do it right. I hope this article can be your road map should you ever need to go there.

A kiss for the Cup.

Blake Couch - Blake is a 25-year veteran of the computing wars who can be taught a new trick or two, but hasn't forgotten paper tape, punch cards, or Teletype machines. Blake resides in Colorado with his wife, daughter, and dog, Stanley Pup, who came along right after the Avalanche last captured Lord Stanley's trophy. Check out Blake's infamous photo of "kissing the cup" compliments of his wife's quick shutter fingers. Blake may be contacted at **Blake.Couch@ERPtips.com.**

January 2006 Volume II Issue 1

ORAtips Journal