DBMaster

Reference Guide for Oracle 11g Migration to DBMaster 5.1

Version: 01.00

Author: DBMaster Support Team and Research & Development Division, Syscom Computer Engineering CO.
Document No: 51/DBM51-T07302010-03-MROR

Publication Date: July 30, 2010

Content

1. Overview

This document is meant to help users successfully migrate their Oracle RDBMS System to DBMaster. This migration process includes not only the schema transition, but also DML, Data Storage.

Oracle is known as the largest RDBMS product in the industry at present. However many of its features such as enterprise usage are sometime regarded very inadequate in most small companies. It is very important for IT professionals to choose the appropriate features. With this document, the users would easily understand the pros and cons between Oracle and DBMaster. Users would be also aware of the characteristics of both DBMaster and Oracle.

In addition, this document could be considered as a reference for people who are already familiar with Oracle, but unfamiliar with DBMaster. It will be very easy to catch the similar idea from DBMaster that they had already known in Oracle. It will shorten the duration of learning curve.

We will introduce DBMaster in the following aspects:

- Migrate Oracle11g database to DBMaster5.1
- Create ANSI-compliant names.
- Customize users, tables, indexes, and tablespaces.
- Remove and rename database objects if they are reserved words in DBMaster.
- To migrate groups, users, tables, primary keys, foreign keys, unique constraints, indexes, rules, check constraints, views, triggers, stored procedures, user-defined types, and privileges to DBMaster.
- Customize the default data type mapping rules.

2. Analyze the current system

We should analyze the system before migrating it from Oracle to DBMaster in some aspects, through which we can evaluate the workloads and costs of the migration.

For instance, we should analyze current operating system and get to know what system we use Windows or any other. Different operating systems have different characteristics. We also need to know what should be considered as emphasis and difficulty in the migration process.

System analyses include both AP system analyses and DB system analyses. In the following chapters we will introduce them in two aspects.

2.1 Analyze AP system

Users should understand system architectures first and know what technologies have been used. Such as Hiberanate, Nhibernate, C, C++, Java, .Net, PHP, Ruby, etc.

In addition, the driver type is also important; users should know which one was used. For example: JDBC, ODBC, DCI, OLEDB, and so on.

Next, analyze the hierarchical structure of AP system, for example: Client/Server, Browser/Server, N-Tier.

Last, users need to analyze the special feature of Oracle and get to know how to convert them into DBMaster. For the special feature, we should consider the following aspects before migration.

- Special features of Oracle
- The workaround of Oracle special feature
- Special syntax of Oracle
- How to convert special syntax into DBMaster

2.2 Analyze Database Objects

For a database, we should analyze all database objects. First of all, we have to know how many database objects should be migrated, for example: tables, views, trigger, etc. We need evaluate how much space is required for storing data.

Next, we should analyze all tables' structures and get to know what contents of these tables will be stored. It can help us divide tables into different table spaces to improve performance. Then users can begin preparing for creating a corresponding database with DBMaster.

There are many differences between Oracle and DBMaster. So we should consider these differences in advance. We will introduce some aspects as followings:

- Data types belong to Oracle but not apply to DBMaster.
- Data types belong to DBMaster but not apply to Oracle.

- Built-functions belong to Oracle but not apply to DBMaster.
- Built-functions belong to DBMaster but not apply to Oracle.
- Indexes belong to Oracle but not apply to DBMaster.
- Indexes belong to DBMaster but not apply to Oracle.

In addition, users can evaluate workloads with above analyses. It's helpful for customers estimating the costs of migration.

3. Setup migration environment

We mainly introduce environment and which aspects users should pay high attention to in this section.

We must ensure the application can run normally with Oracle before we do any works. Then we will install DBMaster and create a database. Certainly, before that users should reserve enough disk space for DBMaster database db files. For convenience, you can install DBMaster in same machine with Oracle.

Next, we need to adjust or configure web server if users' system has web server that is used for deployment and testing web applications.

We also need to pay attention to the following aspects before migrating a database from Oracle to DBMaster.

- Adjust DBMaster configure parameters if necessary.
- Enroll settings in windows system.
- Special workaround for migration.
- DSN or environment variables should be installed.
  - UnixODBC in Linux system (if users move to Linux system)
  - ODBC Driver Manager in Windows
- Which DBMaster. (normal or bundle).
- Which DBMaster server is running? (dmserver or dmservice)
  - dmservice is only for windows system, customers can install it as a windows service with **JServer Manager Tool** or dmsvcutl.exe. Then, the user can set the database service as Auto Start when OS being started. .

4. Methods for migrating table schema and data

Database Migration involves all of the database objects. But we only introduce the migration methods for the table schema and data in this section and other aspects will be described in *chapter 6* and *chapter 7*.

4.1 Database transfer tools

4.1.1 JDATATRANSFER TOOL IN DBMASTER

DBMaster offers users a tool - **JDATA Transfer Tool** for migrating from a third-party database to DBMaster. The **Data Transfer Tool** provides a user-friendly interface for transferring data in and out of the database. The tool performs the following functions:

- Import from text
- Import from XML file
- Import from ODBC
- Export to text
- Export to XML
- Batch transfer

For more information about performing each type of data transformation please reference *JDBA Tool Chapter Data Transfer***.**

Here we mainly introduces **Import from ODBC** . DBMaster supports importing data from other data sources via ODBC. Other data sources may include other database engines, such as Oracle, Microsoft SQL Server, etc.

A large number of software applications have been developed to be compatible with Open Database Connectivity (ODBC). ODBC is an industry standard for sharing data among diverse data sources. DBMaster can import data from any ODBC compliant data sources through the **Import from ODBC wizard**.

Data may be imported using the flowing three methods:

- Choose the tables directly
- With one or more SQL SELECT statements
- Via an XML batch file

Furthermore, you may specify the mapping of column data through the transformation function. The transformation function supports direct column-to-column mapping or mapping through SQL

SELECT and SQL INSERT statements. When importing data directly from tables or through SQL SELECT statements which allow saving a 'map' of the data transformation to an XML batch file. The XML batch files are saved as a well-formed XML document with a form that can be parsed by the **Data Transfer Tool**. Batch files can be used to import table schema from a data source to multiple DBMaster database.

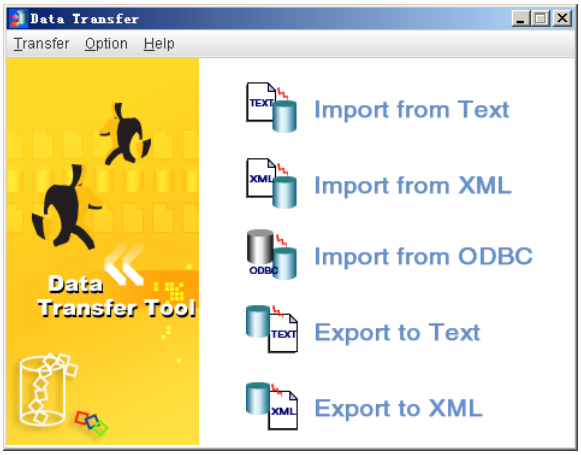
4.1.1.1 How to start the DBMaster JData Transfer

The **Data Transfer Tool** is a separate application which can be started as GUI.

Start>**programs**>**DBMaster 5.1**>**DataTransfer**, or start within **JDBA Tool**.

4.1.1.2 Execute steps Import from ODBC

Step 1: Open the **Data Transfer Tool**.



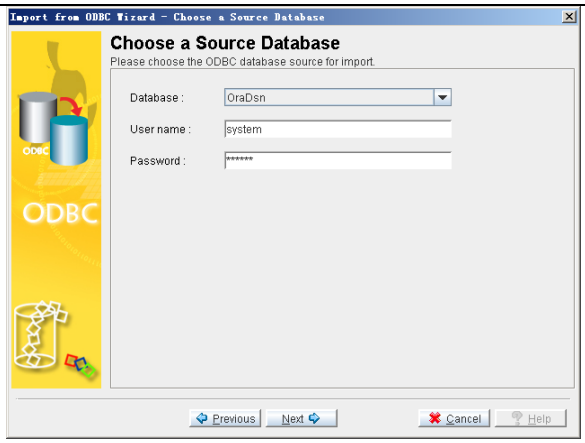
Step 2: selected **Import from ODBC** option and open the **Import from ODBC Wizard**.

Step3: Click on **Next**. The **Choose a Source Database** window appears.

Database: Select the DSN name in the Database drop-down list.

Username: Enter a user name into the appropriate field.

Password: Enter corresponding password into the appropriate field.

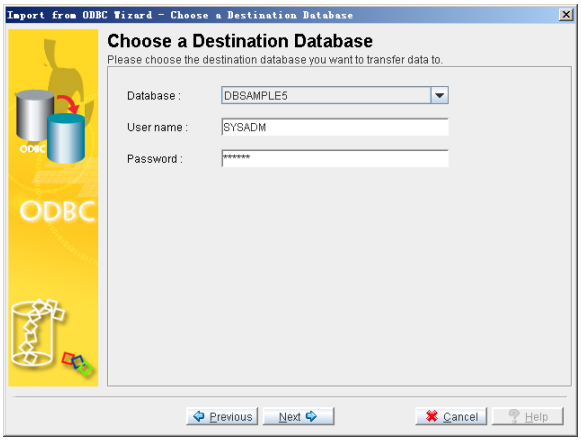


Step 4: Click on **Next**. The **Choose a Destination Data Source** window appears.

Database: Select the DSN name in the Database drop-down list.

Username: Enter a user name into the appropriate field.

Password: Enter corresponding password into the appropriate field.



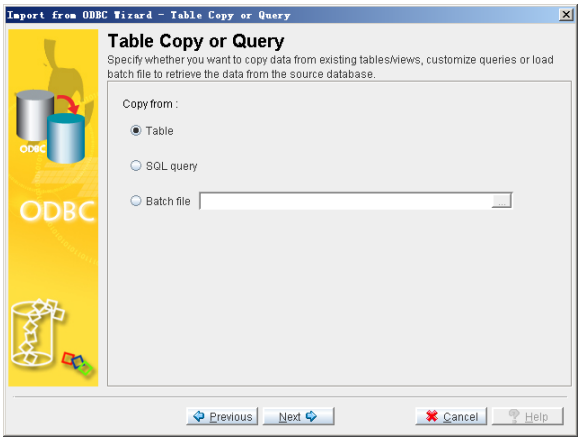
Step 5: Click on **Next**. The **Table Copy or Query** window appears.

There are three provided options. Select one of the three methods for data transfer:

Table: To import data from a list of tables,

SQL query: To import data using a series of SQL SELECT statements

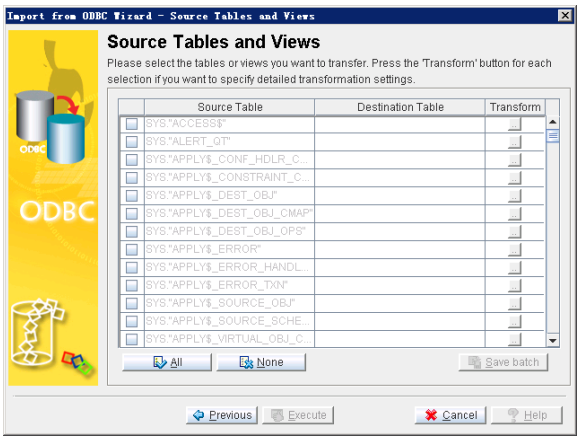
Batch file: To import data through an XML file.



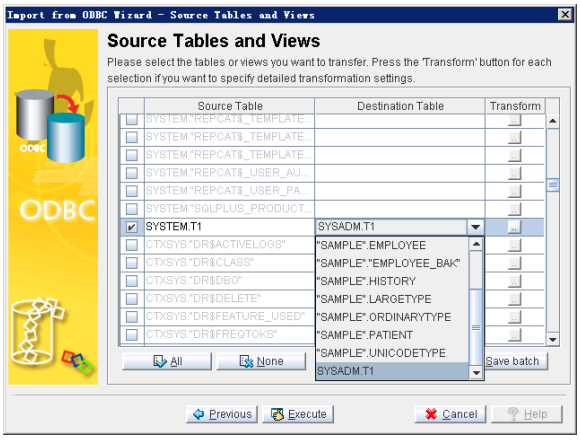
Three choices and corresponding operations:

➢ Selected" Table" check box

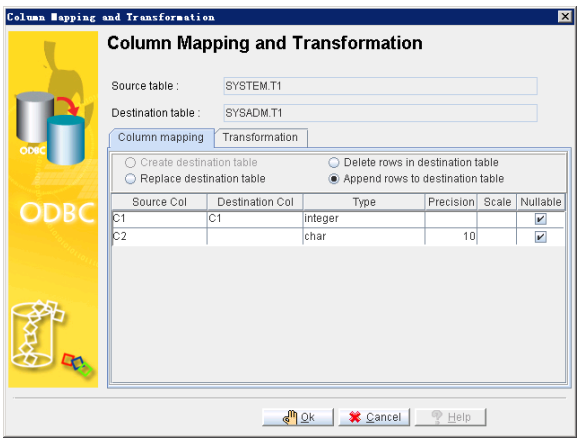
Sub_step 1: Click on **Next**. The **Source Tables and Views** window appears. All tables from the source database will appear in the Source Table column. Check the box to the left of each table to import.



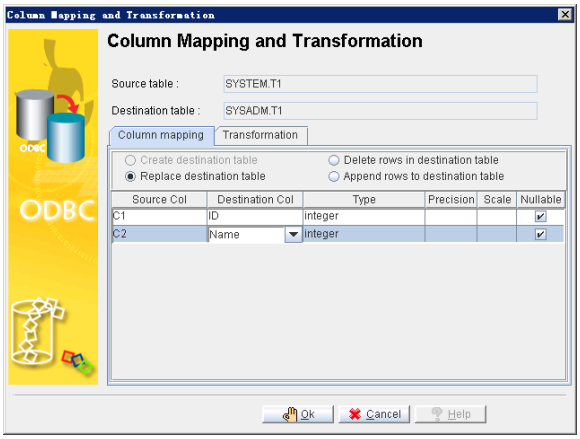
Sub_step 2: For each source table or view selected, click on the **Destination Table** field. If desired, change the name of the destination table by selecting a new table from the menu or entering a new name.



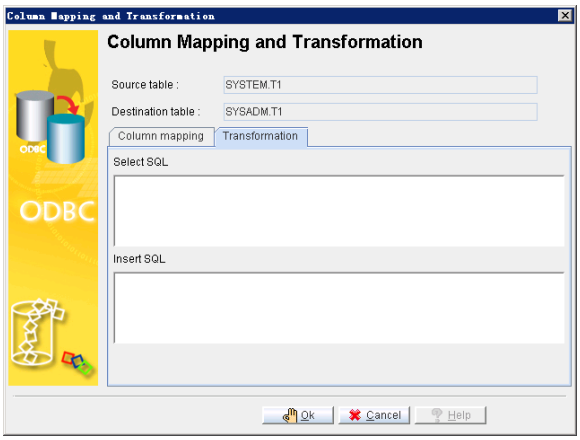
Sub_step 3: You can modify **Column mapping** or the result set to import by clicking on the **Transformation** button of the corresponding source and destination table.



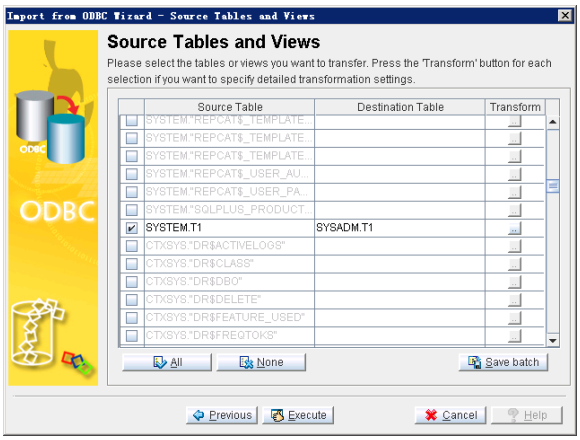
Sub_step 4: Change the name of the destination column by selecting a new column from the menu or entering a new name.



Sub_step 5: Click on the **Transformation** tab to specify constraints on the result set. Enter a Valid SQL SELECT statement into the **Select SQL** field and a valid SQL INSERT statement into the **Insert SQL** field.

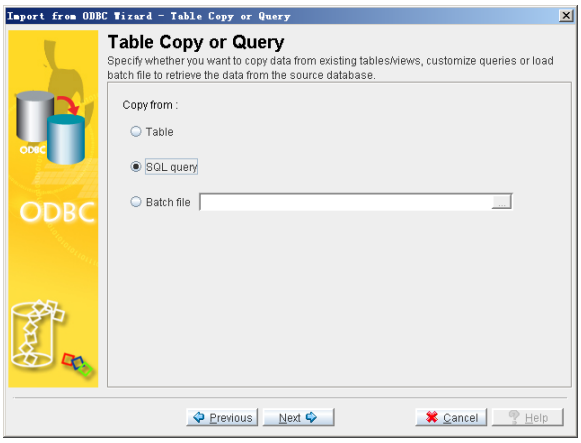


Sub_step 6: Click on **OK** to return to the **Source Tables and Views** window. You may also choose to save the map of the import ODBC schema to an XML file by clicking on **save batch**.

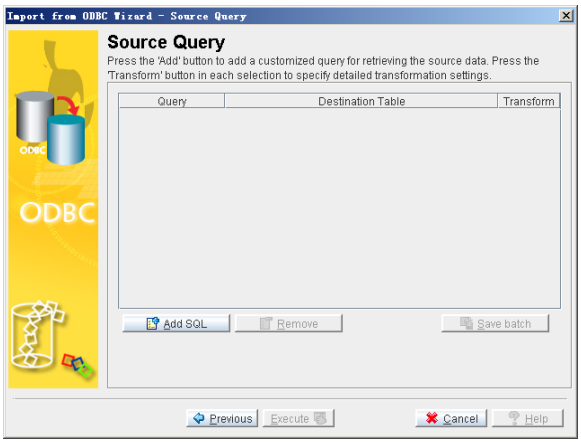


- **Select “SQL query” check box**

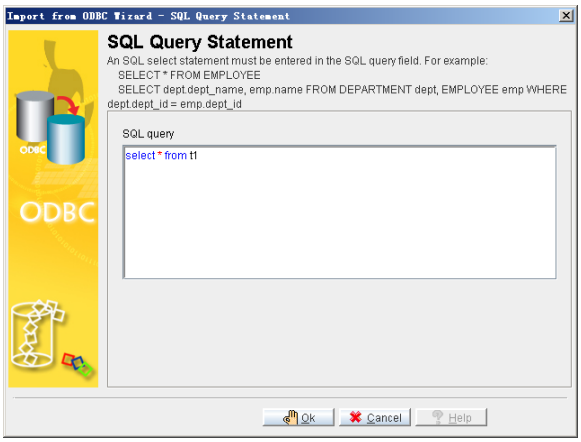
Sub_step 1: Click on **Next**. The **Table Copy or Query** window appears. Click on **Add SQL**. The **SQL Query Statement** window appears. And enter a valid SQL SELECT statement into the **SQL Query** field.



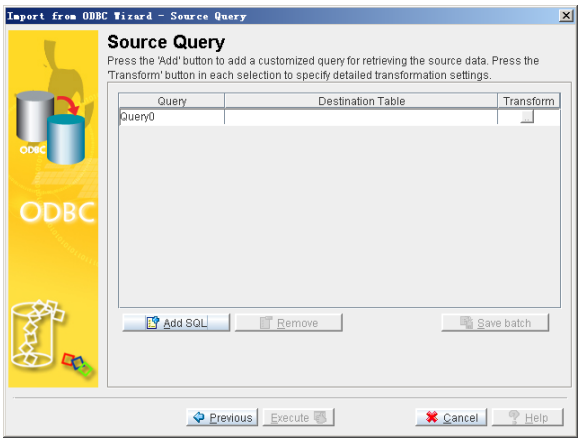
Sub_step 2: Click on **OK**. The **Source Query** window reappears.



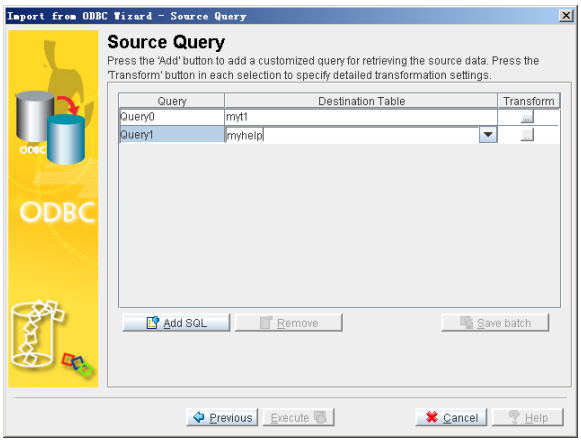
Sub_step 3: Click on **Add SQL**. The **SQL Query Statement** window appears. Enter a valid SQL SELECT statement into the **SQL Query** field.



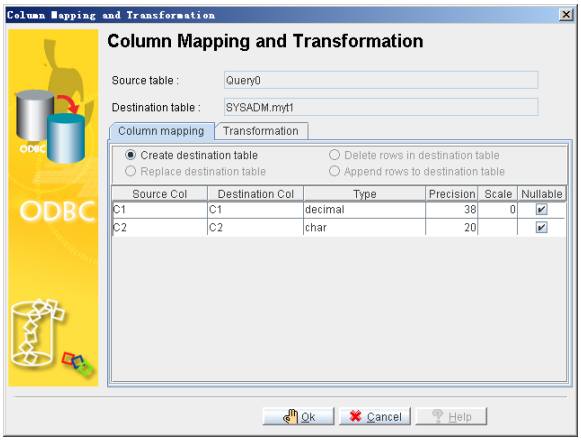
Sup_step 4: click on **OK** to return to **Source Query** page.



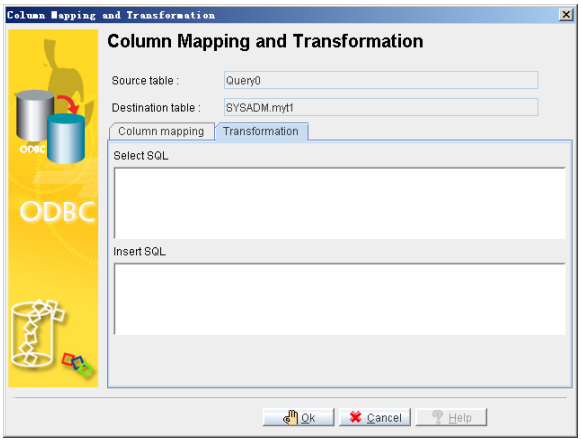
Sub_step 5: You can add more SQL query statements by clicking on **Add SQL** and change the name of the destination column by selecting a new column from the menu or entering a new name.



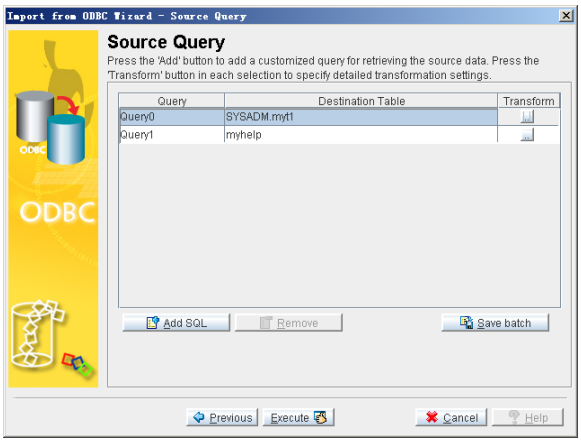
Sub_step 6: you also can modify the mapping of source and destination columns by clicking on the **Transformation** button.



Sub_step 7: Click on the **Transformation** tab to specify constraints on the result set. Enter a Valid SQL SELECT statement into the **Select SQL** field and a valid SQL INSERT statement into the **Insert SQL** field.

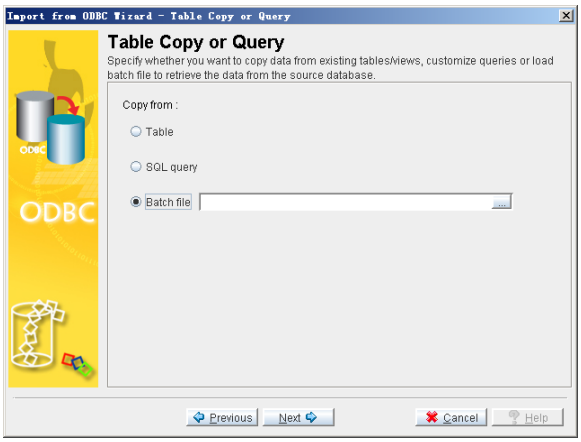


Sub_step 8: click on OK and return to **Source Query** page. You also can choose to save the map of the import ODBC schema to an XML file by clicking on **Save batch**. The Save Batch File will open. Select or create an XML file to save the imported ODBC map schema. Click on **Save Batch File** to create the XML file.

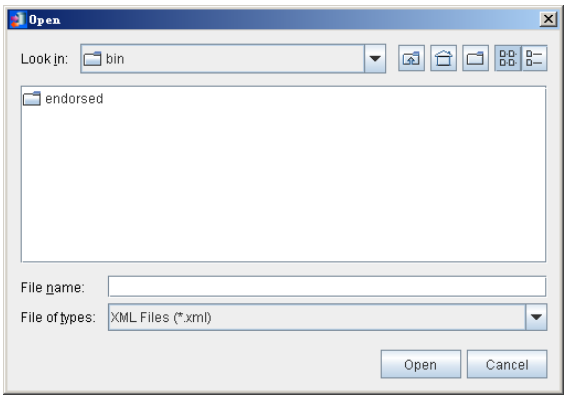


➢ Select “Batch file” Check box

Sub_step 1: Select an XML file from which to import the ODBC map schema. Click on **Open**. The **Table Copy or Query** window reappears.

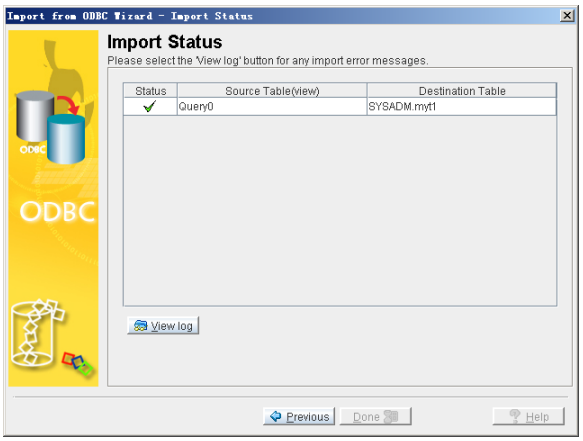


Sub_step 2: Click on **Next**. The **Source Query** window will open, displaying a mapping Schema according to the XML file.

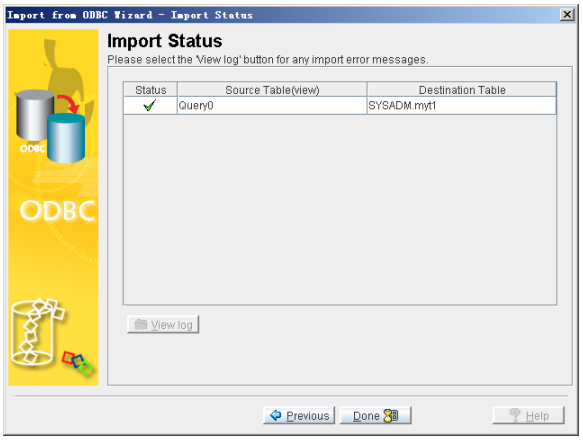


Note: The remaining operations are same as select **Table.**

Step 5: Click on **Execute** to import the source data. The **Import Status** window appears.



Step 6: If errors appear, click on **View log** and scroll to the bottom to see the error message. If no errors occurred, click on **done**.



4.1.2 ORACLE SQL DEVELOPER TOOL

Oracle **SQL Developer** is an intuitive tool that enables you migrate source database objects to the destination database. The **SQL Developer** installation is very simply .you should only do a full install with Oracle Database 11g Release installation which ships with **SQL Developer1.1.3**. As with other installs, create a new folder and unzip the latest download installation for Oracle **SQL Developer**.

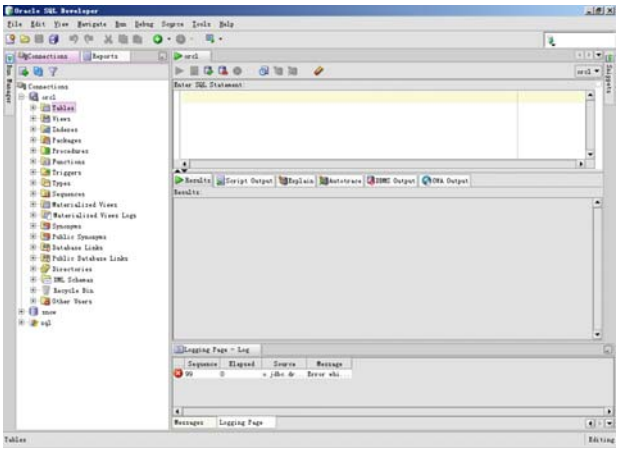
The choice of file formats of exporting the data is (version: 1.1.3):

- CSV—comma separated values
- HTML—HTML tagged texts
- Insert—SQL DML commands
- Loader—SQL Loader file formats
- Text—unstructured texts
- XLS— Microsoft Excel spreadsheets
- XML—XML tagged texts

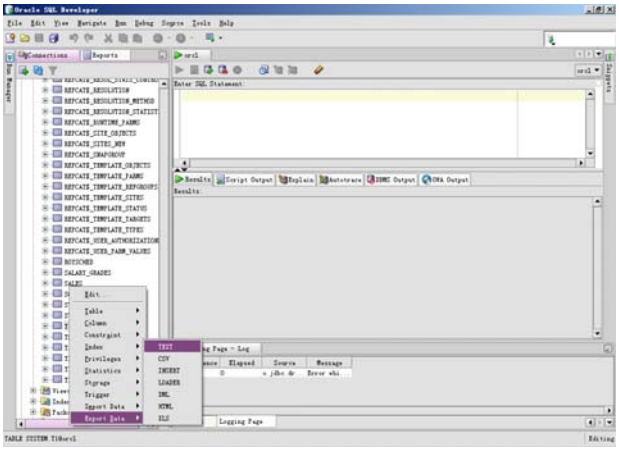
We need two processes for migration. One is exporting data from Oracle by **SQL Developer** and the other is importing data into DBMaster by **JDATA Transfer Tool**.

4.1.2.1 Export Data from Oracle

Step 1: Start the tool Oracle **SQL Developer** by **Start**>**programs**>**Oracle_OraDb11g_home**>**application program develop** >**SQL Developer**

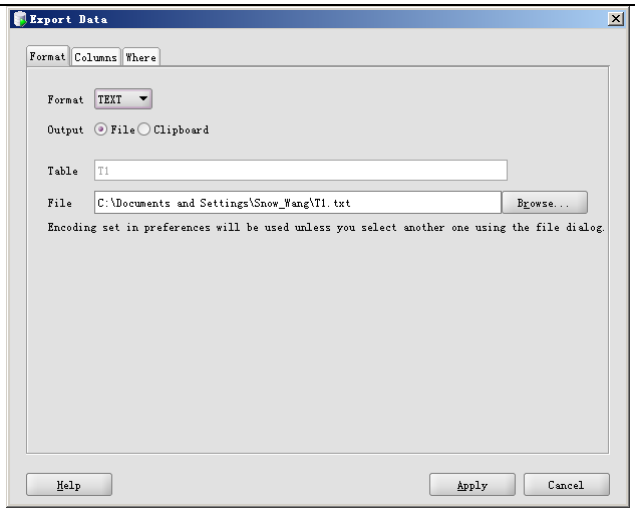


Step 2: Invoke export data by right-click on a particular table which you want to export the Connections navigator and chose an appropriate output format.

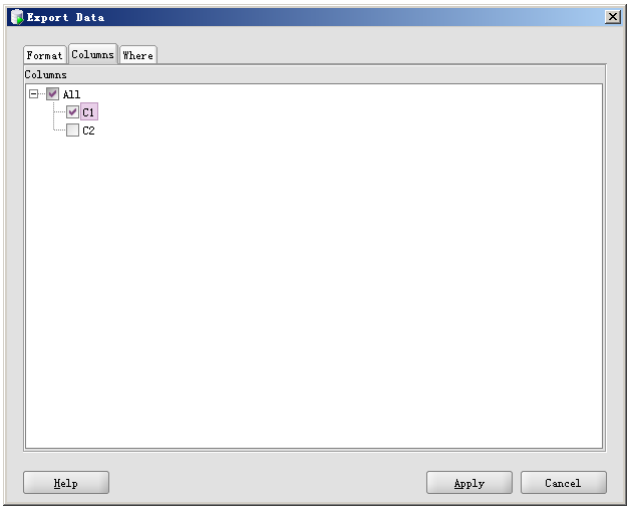


Step 3: There are three tabs in this Wizard: **Format**, **Columns**, **Where**

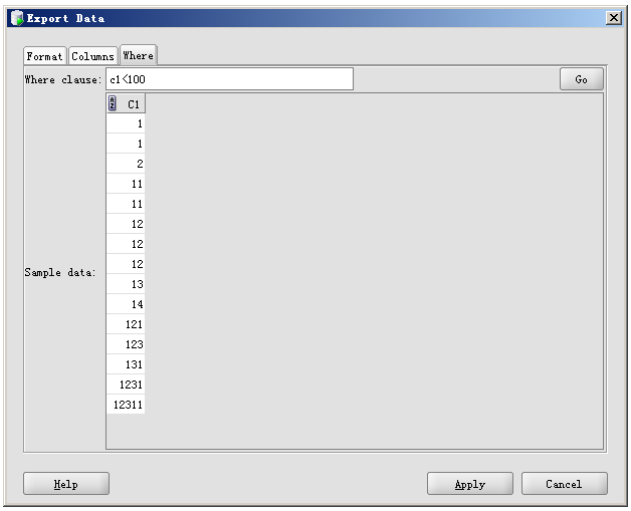
Format: this step you can mainly change output file format from the dropdown list and set the export file location by entering the full path or click on the **Browse** button to search.



Columns: you can select the columns that you want to export. Once you have selected the columns, you can also restrict the returned data. Add the WHERE CLAUSE restriction criteria and click on **Go** to restrict the returned data



Where: you can modify the number of returned rows by including the Where clause, the query is only re-run on clicking on **Go**, and not by clicking on **Apply**, which merely writes the data to files.



Step 4: Click on **Apply** and output files will be saved in folders which are specified by **step 3.**

4.1.2.2 Import Data to DBMaster

Import data from appropriate format files which are exported from Oracle. Currently **JDATA Transfer** supports three kinds of formats for importing: TEXT, XML and ODBC. So we should select the format both **JDATA Transfer** and **SQL Developer** can support. We will introduce the other two import methods in this section.

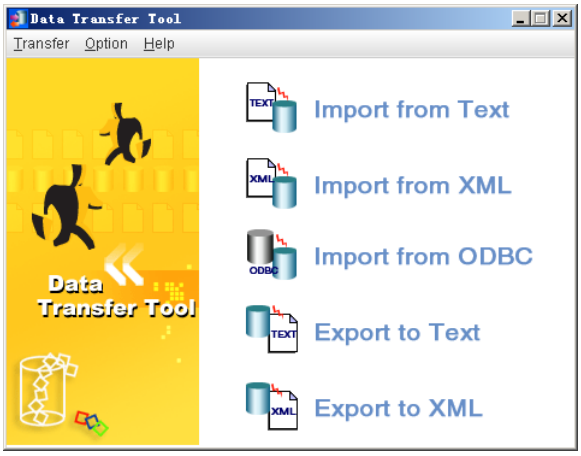
4.1.2.2.1 Import from TEXT

The ability to import table data from a text file is an important feature in a database, and is made easy with the **Data Transfer Tool**. Text data must be properly formatted to be acceptable for importing. Data may be imported to the database only from a properly formatted text file.

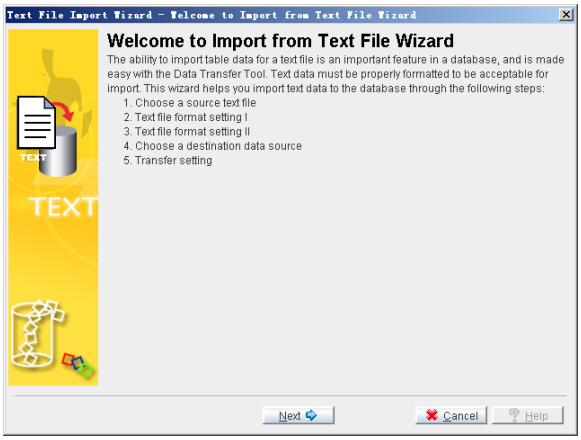
Before attempting to import data from a text file, you should check the format of the output file that you want to import. Some important settings to consider the format of a text file include: Row Delimiter, Column Delimiter, Text Qualifier, Binary Qualifier, and so on.

- **Importing a text file to a database:**

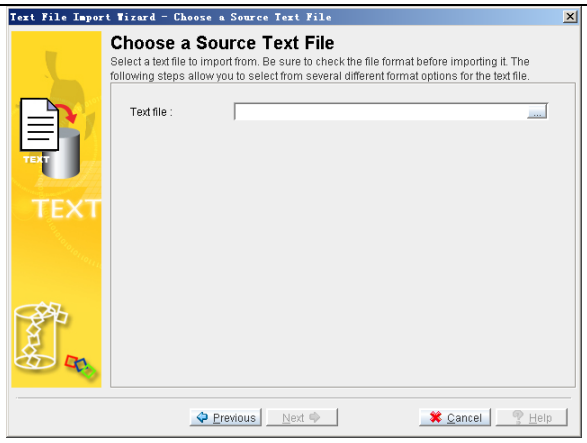
Step 1: Open the **Data Transfer Tool**.



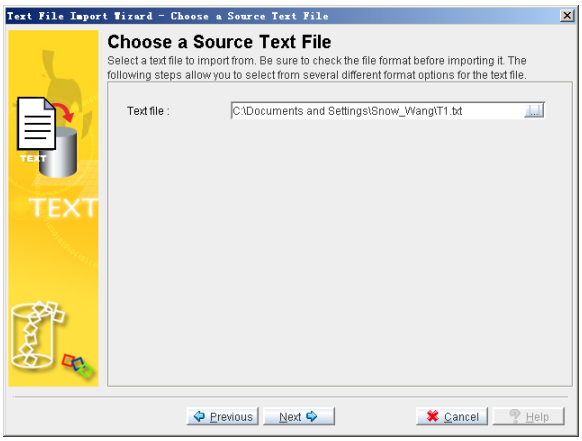
Step 2: Select **Import from Text** from the Transfer menu. The **Welcome to Import from Text File Wizard** window will open, displaying a summary of the steps to be taken in the wizard.



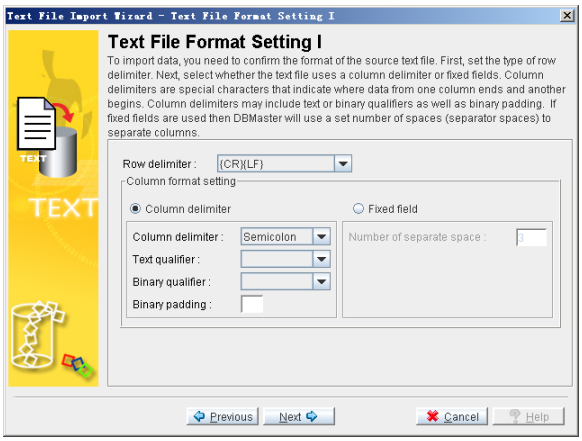
Step 3: Click on **Next,** the **Chose a Source Text File** window appears.



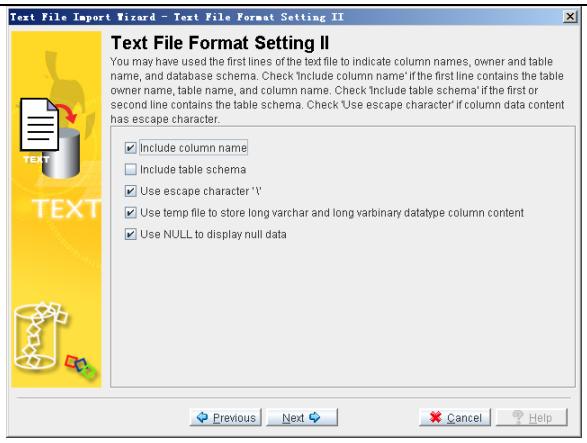
Step 4: Enter the full path of a text file to import or click on the **Browse** button to search for a text file.



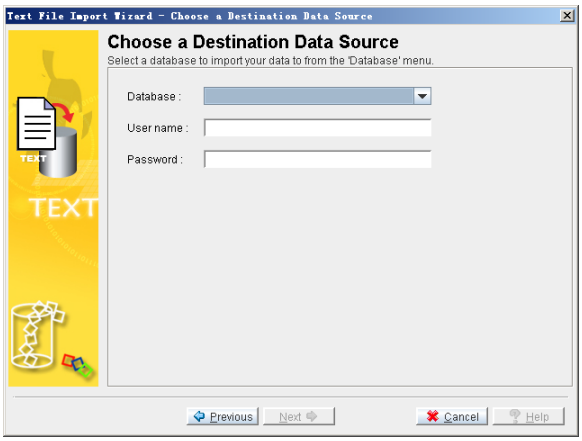
Step5: After you have selected a text file, click on **Next**, the **Text File Format Setting I** Window appears. Open the text file in a text editor to check the format of the data and select the appropriate settings for the format of the text file you are importing.



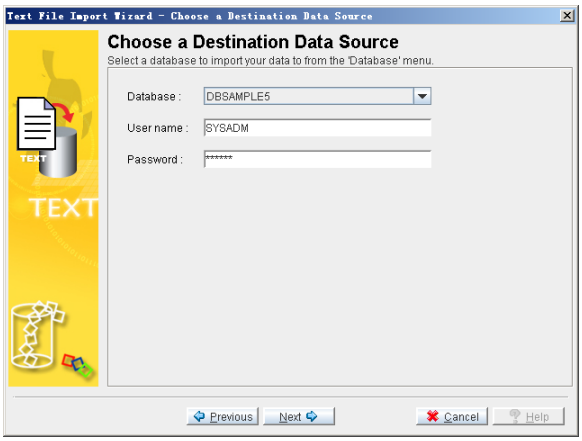
Step 6: Click on **Next**, the **Text File Format Setting II** window appears



Step 7: Finish selecting the appropriate settings for the format of the text file you are importing. Click on the **Next** button.

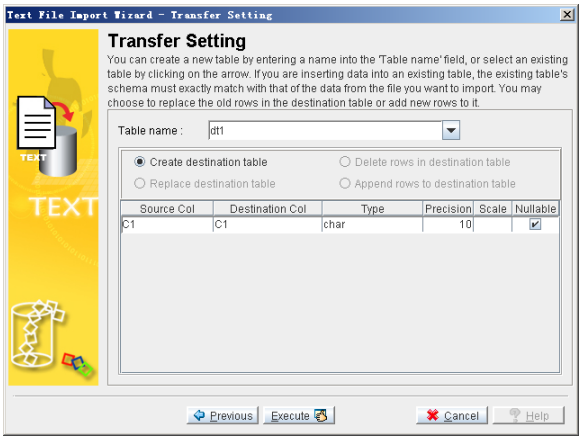


Step8: Select the database to import data to from the Database menu and enter a user name and the password into the appropriate fields

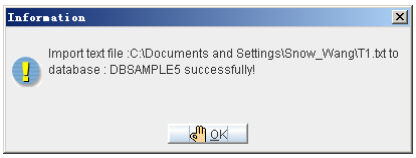


Step 9: Click on **Next.** The **Transfer Setting** window appears.

Step 10: Enter a new table name into the **Table Name** field, or select a table from the menu. Selecting a table from the menu will allow you to choose to replace the destination table, delete rows in the destination table, or append new rows to the destination table.



Step 11: Click on **Execute** to import the text file. A confirmation dialog box appears.



Step 12: Click on the **OK** button. Check data and ensure data has been migrated to DBMaster successfully.

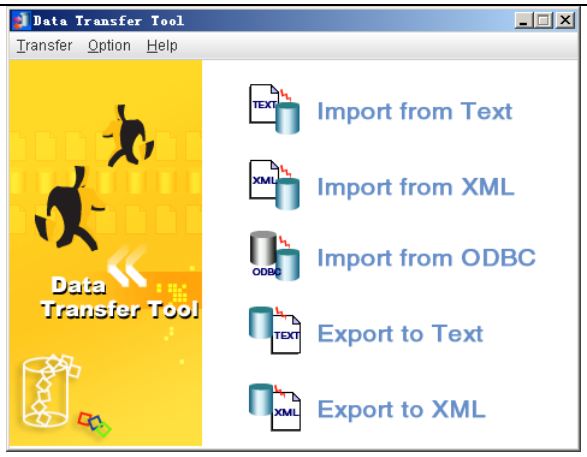
4.1.2.2.2 Import from XML

XML files can be imported into the database also. XML tags should be defined first in a Document Type Definition (DTD) file before being imported into the database. Furthermore, the DTD may define the schema in a way that is acceptable to the database.

It is important to consider the structure of the XML file you wish to import to DBMaster. To ensure that the structure of the XML file and associated DTD have compatible structure.

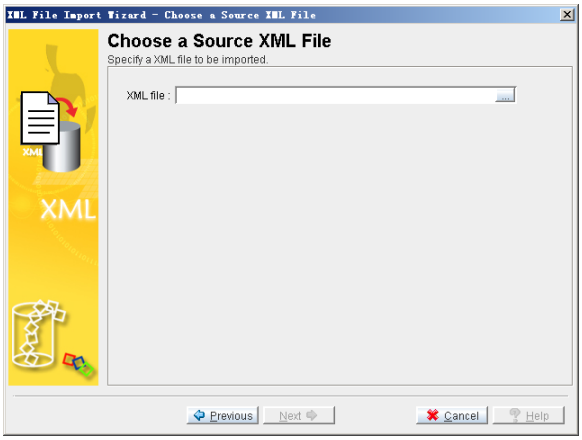
- **Importing Data from an XML file**

Step 1: Open the **Data Transfer Tool**.

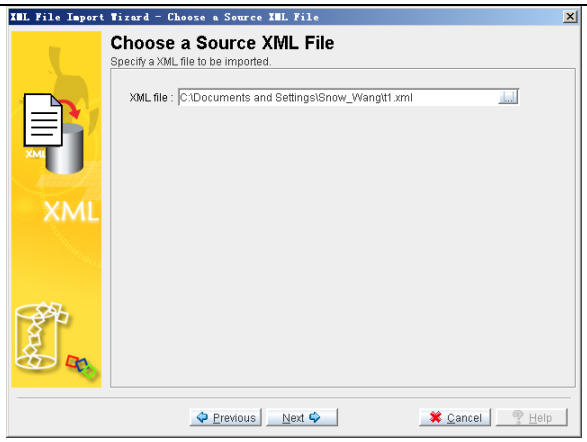


Step 2: Select **Import from XML** from the Transfer menu. The **Welcome to Import from XML File Wizard** window appears.

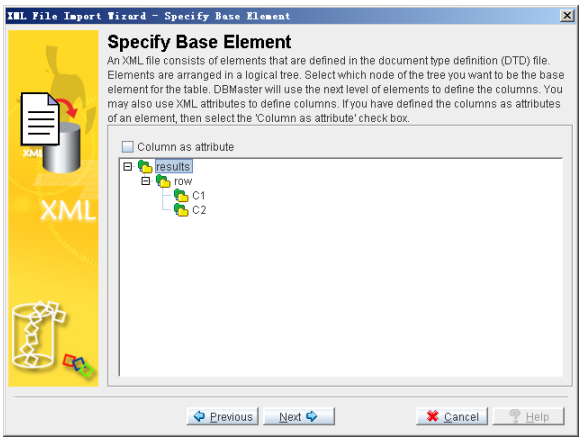
Step 3: Click on **Next.** The **Choose a Source XML File** window appears.



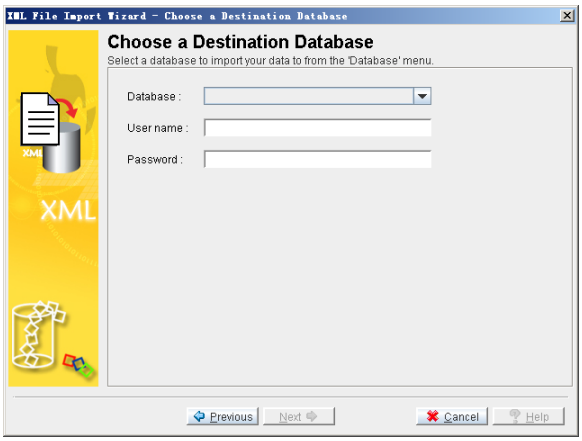
Step 4: Enter the full path of a text file to import or click on the **browse** button to search for a text file.



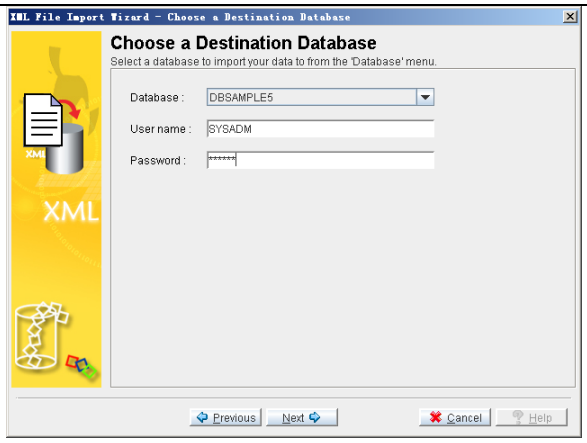
Step 5: Click on **Next**. The **Specify Base Element** window appears. The nodes of the tree structure represent the elements in the XML file. Click on the nodes on the tree until they are fully expanded. Select a parent element to be the table name. The child elements will become the columns of the table. Check Column as attribute if appropriate.



Step 6: Click on **Next**. The **Choose a Destination Data Source** window appears.

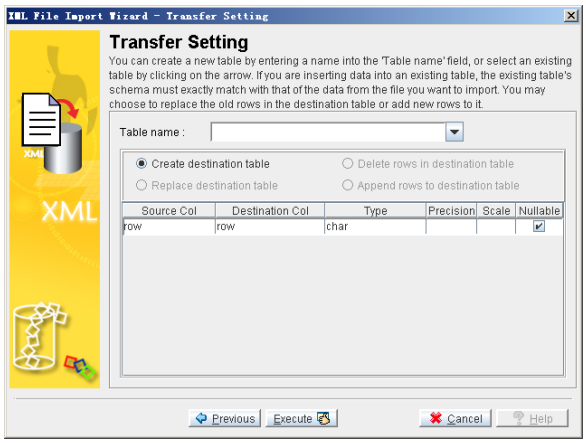


Step 7: Select the database to import data to from the Database menu and enter a user name and the password into the appropriate fields.

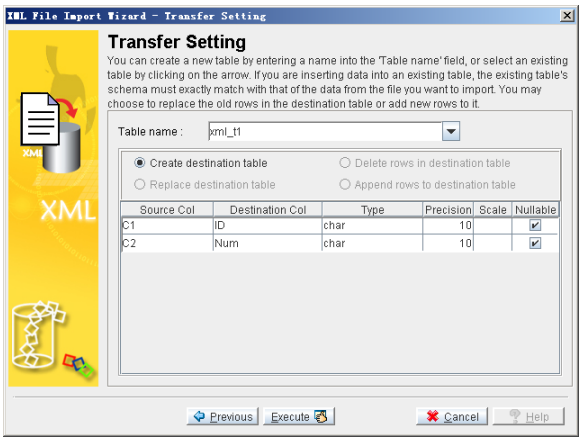


Step 8: Click on **Next.** The **Transfer Setting** window appears

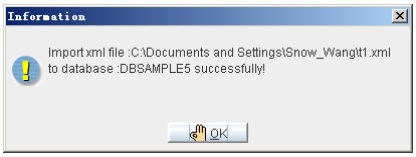
Step 9:



Step 10: Enter a new table name into the **Table Name** field, or select a table from the menu. Selecting a table from the menu will allow you to choose to replace the destination table, delete rows in the destination table, or append new rows to the destination table.



Step 11: Click on **Execute** to import the XML file. A confirmation dialog box appears.



Step 12: Click on the **OK** button. Check data and ensure data had been migrated to DBMaster successfully.

4.2 Other Third party tools

Currently，there are many kinds of database migration tools can be used. Some of them are popular which can be used for most of databases. Certainly, some of them are only designed for special databases.

The user can choose a popular tool for their migration according to different requests.

In following sections, we will introduce two popular database migration tools.

4.2.1 SQL SCRIPT BUILDER

SQL Script Builder is a powerful software by which users can create a database migration sql script (or dump file) or database files from any ODBC data source. The script will migrate the database (multiple tables selection) or only one table and records in it. Scripts are available in five output formats; MySQL, MS SQL, Oracle, Pervasive and PostgreSQL, and files come in Access mdb, Excel csv, MS xml. **SQL Script Builder** is very simple to use, you just have to choose the database and tables from the list. SQL Script Builder scripts can be used on your DBMS (database management system) or uploaded on a server.

SQL Script Builder can be used. For example, if you migrate a database from Oracle database to DBMaster, you don't have to transfer whole database, you can import only one table at a time and have no limit, what you need is the ODBC driver for the database you wish to import from. ODBC is a universal interface, almost every database provider supports it.

With **SQL Script Builder,** you can create an ODBC connection for origination database and generate the script, then, you need to ensure the script can work well on the destination database.

4.2.1.1 Migration methods

We need two steps for an integrity migration from Oracle to DBMaster. One is exporting data from Oracle by **SQL Script Builder** and the other is importing data in DBMaster by **JDATA Transfer Tool**.

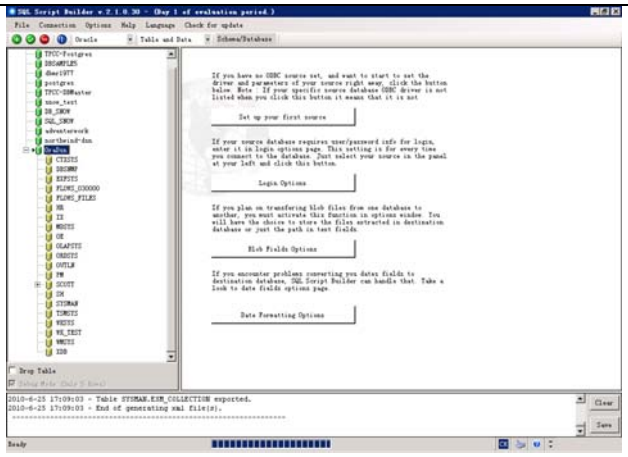
First, Use this tool to convert the data from the origination database to a supported file format. For DBMaster, we recommend the XML format.

Then, Use **JDATA Transfer Tool** in DBMaster and select **Import from XML** option to import datafiles which have been exported from Oracle.

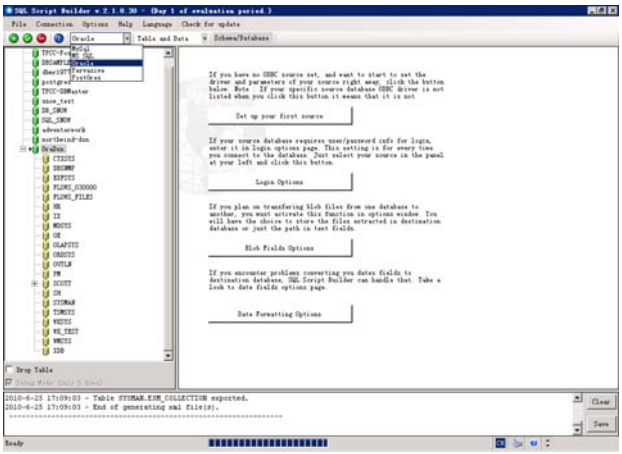
4.2.1.2 Migration steps

- **The simply operation steps for exporting XML file with SQL Script Builder**

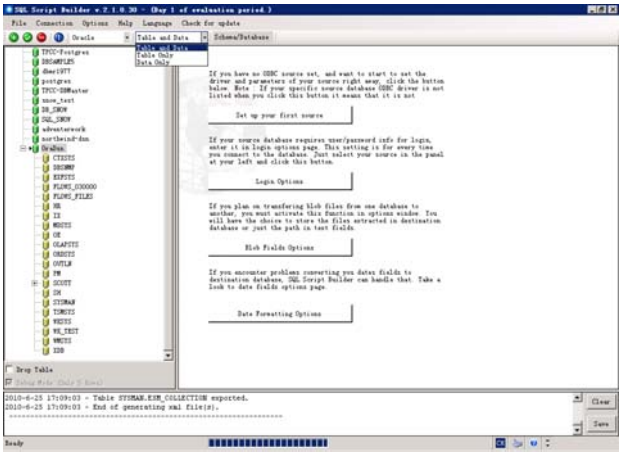
Step 1: Open **SQL Script Builder**.



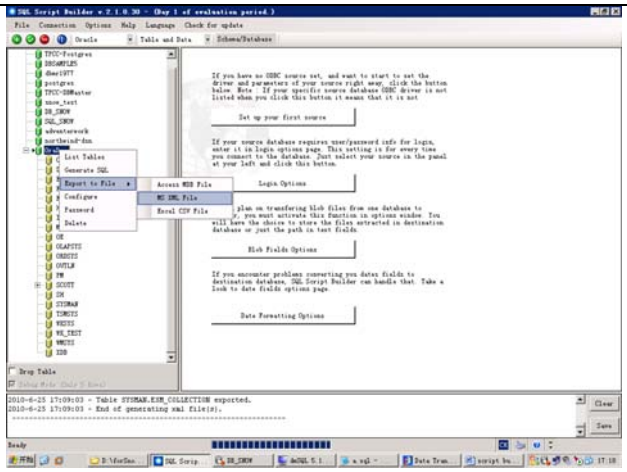
Step 2: Select the source database type from dropdown-list.



Step 3: Select migration data contents from the dropdown-list.

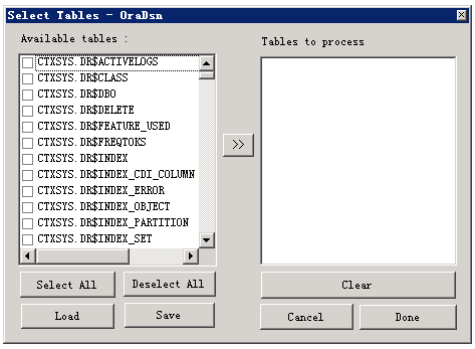


Step 4: Select XML file formats.

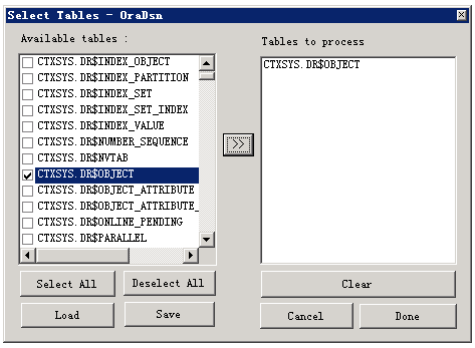


Step 5: Select export XML files location.

Step 6: Display source tables.



Step 7: Select tables that you want to migrate to DBMaster.



Step 8: XML files had been exported from Oracle successfully.

- **The simply operation steps for Import XML file with JDATA Transfer**

More details of importing from XML please refers to *chapter 4.1.2.2.2*.

4.2.2 ORALOADER TOOL

OraLoader is an Oracle data load/unload tool. **OraLoader** allows you to proceed quickly and efficiently in your Oracle data load/unload work. You can use it to migrate data from Oracle to DBMaster easily.

The main Export features are:

- Export data to TEXT, CSV, EXCEL, HTML, XML, SQL*Loader control files

- Export all tables data in a schema to files in the same time
- Run SQL Commands and export results to TEXT, CSV, EXCEL, HTML, and XML.

4.2.2.1 Migration methods

We need two steps for an integrity migration from Oracle to DBMaster. One is exporting data from Oracle by **OraLoader** and the other is importing data in DBMaster by **JDATA Transfer Tool**.

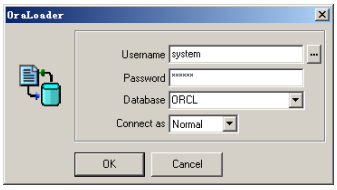
First, Use this tool to convert the data from the origination database to a supported file format. For DBMaster, we recommend the XML or TEXT format. Here we select TEXT format as a sample.

Then, Use **JDATATransfer Tool** in DBMaster and select **Import from XML** or **Import from TEXT** option to import datafiles which have been exported from Oracle.

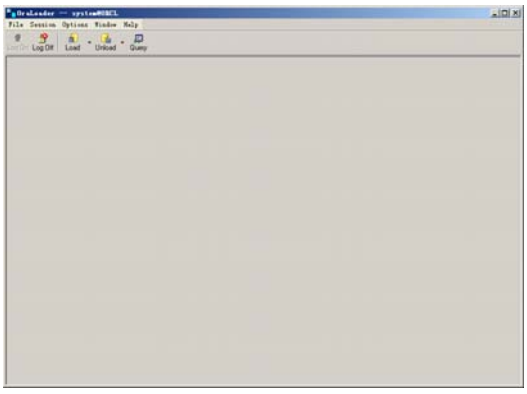
4.2.2.2 Migration steps

- **The simply operation steps for Export TEXT file with OraLoader**

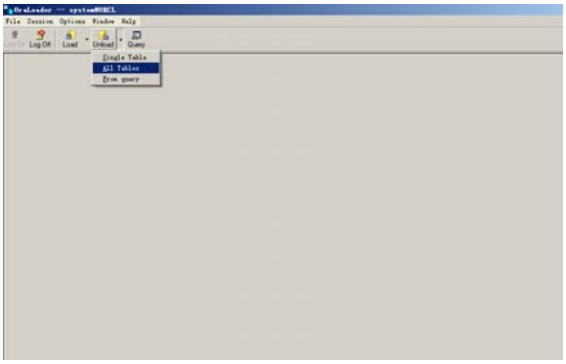
Step 1: Start>**programs**>**OraLoader** and enter correct username, password, chose source database, connect mode from the drop-down list.



Step 2: Click on **OK** button and open **OraLoader**

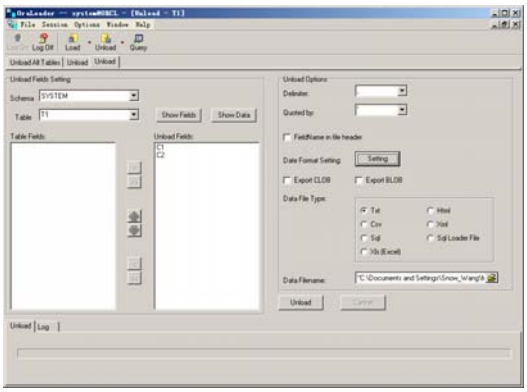


Step 3: Click on **Unload** menu, there are three provided options: **All Tables, Single Table** and **From query.** You can choose one of them according to your need. Here we select **All Tables** as a sample**.**

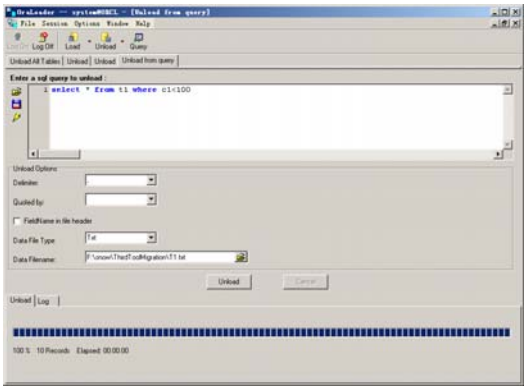


Note:

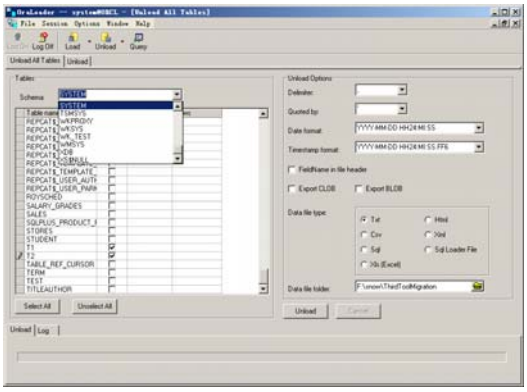
1) If you select **Single Table** option then show the Unload Single Table form. You can select schema and table, you also can select needed column to export in this page. One table can be unloaded one time.



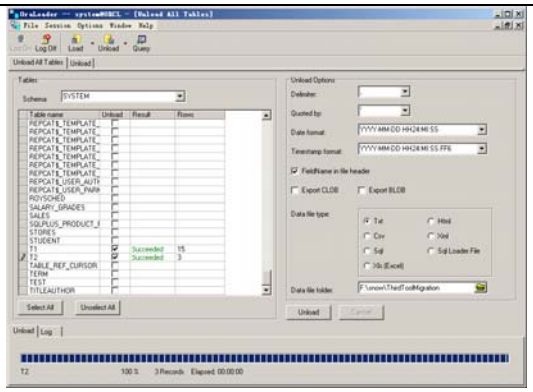
2) If you select **From query** option then show the query form and run sql.



Step 4: All tables will be show. Select schema, tables and configure Unload options in this page. You can select tables which you want to unload by selecting **Check Box** and configuring **Unload** Options on right panel that include selecting appropriate **Delimiter**, **Quoted by**, **Date format**, **Timestamp format** from the drop-down list, changing **FileldName in file header**, **Export CLOB**, **Export BLOB** status by Check Box, selecting Data file types by radio-button and selecting Data file folder location by clicking on **Brower.**



Step 5: Click on **Unload** button, and show unload outputs.



Step 6: Table T1 and T2 had been unloaded with the TEXT format. We can check the export files in the Data file folder specified location.

- **The simply operation steps for Import TEXT file with JDATA Transfer**

More details of Importing from TEXT please refers to *chapter 4.1.2.2.1*.

4.3 Modify DDL manually

If you are familiar with DDL, you can export all the schema and data with the TXT format from Oracle firstly. And modify the schema and make the syntax and data types fit to DBMaster. For example: Via **Developer Tool** (Tools/Export DDL (and Data))

Then, run the schema script in **dmSQL** tool or **JSQL** tool. Please modify and try again if any error occurs.

Certainly, if there are many errors or the data amount is huge, you can only export the DDL (not include data) firstly. Then modifying and successfully running DDL files in DBMaster. Last export the data from Oracle with TXT or XML file format which described in *chapter 4.1.2.1*. And import into DBMaster via **JDATA Transfer Tool** which described in *chapter 4.1.2.2*.

Note: If you don't want to use **JDATA Transfer Tool** to import, after exporting the data to TXT format files, you may modify them for DBMaster and run in **dmSQL** or **JSQL tool**, or even use the import command.

4.4 Write code

Users can use a programming language they are familiar with to develop a simple script or tool for migrating databases. The work theory and process steps are similar with above manual methods.

5. Compare Oracle and DBMaster

5.1 Schema Comparison

5.1.1 THE TERMINOLOGY COMPARISON

The following table enlists the terminologies in DBMaster and Oracle. In many aspects, DBMaster have more common characteristics with Oracle than SQL Server. However, there are some differences between them.

Oracle	DBMaster
Database	Database
Schema	Database user or owner
Tablespace	Tablespace
Segment	N/A
Extent	N/A
Block	Page/Frame
User	User
Role	Group
Table	Table
Partitioned tables	N/A DBMaster supported after 5.2 version
View	View
Temporary tables	Temporary tables
Cluster	N/A
Check constraint	Check constraint
Sequences	Serial
Synonyms	Synonyms
Triggers	Triggers
Column default	Column default
Unique key	Unique index
Primary key	Primary key
Foreign key	Foreign key

Index	Non-unique index
Packages	N/A
PL/SQL Procedure	Embedded-SQL (ESQL/C) stored procedure/ Java stored procedure SQL SP is supported after DBMaster 5.1
PL/SQL Function	ESQL/C stored procedure

5.1.2 STORAGE STRUCTURE COMPARISON

Oracle is aimed at the enterprise usage in many terms, but DBMaster is not meant to dedicate to the same scale as Oracle. In result, some of storage settings or files are not valid in DBMaster. Users should be aware of unsupported storage types in DBMaster. Try to solicit and convert some missing settings or files into the correspondence.

Oracle	DBMaster
Parameter File (initdb.ora)	Configuration File (dmconfig.ini)
Control File	Configuration File (dmconfig.ini) and Catalog (.SDB, .SBB)
Data Files	Data Files
Redo Log	Journal
Rollback Segment	Journal
Temporary Segment	Temporary File (.TMP)
Data Dictionary	Catalog
Snapshot	N/A

5.1.3 PROCESS AND RELATED TERM DEFINITION

In Oracle, every task would be taken care by some specific processes. DBMaster will use the general process instead of individual processes. Therefore, a process in DBMaster will usually comprise many Oracle processes.

Oracle	DBMaster
Instance	Database server
DBWR (Database Writer)	I/O daemon
LGWR (Log Writer)	I/O daemon
CKPT (Checkpoint)	I/O daemon
PMON (Process Monitor)	DBMaster Server
SMON (System Monitor)	DBMaster Server
RECO (Recovery)	DBMaster Server
ARCH (Archiver)	Backup Server

Redo	Replay Journal
Undo	Rollback Journal
Recovery	Recovery
Checkpoint	Checkpoint
Backup	Backup
Restore	Restore
Import	Import/Load
Export	Export/Unload
Archive	Backup
SGA (System Global Area)	DCCA (Data Communication Control Area)
Shared Pool	SCA (System Control Area)

5.1.4 RESERVED WORD CONFLICT IN DATABASE OBJECT

SQL Server and DBMaster reserved words are different. Many DBMaster reserved words are valid object names or column names in Oracle. Likewise, many Oracle reserved words are valid object names in DBMaster. Using of reserved words as database object names makes it impossible to use the same names across the two databases.

Choose a unique database object name by case and by at least one other characteristic, and ensure that the object name is not a reserved word from either database.

Costumers can write object names in double quotation marks in DBMaster if you want to use reserved words as object names. Oracle can do the same treatment to use reserved words.

For example,

In DBMaster: create table test ("ADD" int);

In Oracle: create table test (“ADD” int);

Different from Oracle, in DBMaster, we also can set keyword **DB_ResWd** to be 0 in dmconfig.ini file before database creation, which allows objects containing reserved words to be imported.

For a list of reserved words in DBMaster, see the *SQL basics,* reserved words in **DBMaster 5.1 On Line Help**.

Oracle	DBMaster
ACCESS, ADD, ALL, ALTER, AND, ANY, AS, ASC, AUDIT, BETWEEN, BY, CHAR, CHECK, CLUSTER, COLUMN, COMMENT, COMPRESS, CONNECT, CREATE, CURRENT, DATE, DECIMAL, DEFAULT, DELETE, DESC, DISTINCT, DROP, ELSE, EXCLUSIVE, EXISTS, FILE, FLOAT, FOR, FROM, GRANT, GROUP, HAVING, IDENTIFIED, IMMEDIATE, IN, INCREMENT, INDEX, INITIAL, INSERT, INTEGER, INTERSECT, INTO, IS, LEVEL, LIKE, LOCK, LONG, MAXEXTENTS, MINUS, MLSLABE, MOD, MODIFY, NOAUDIT, NOCOMPRESS, NOT, NOWAIT, NULL, NUMBER, OF, OFFLINE, VALUES, VARCHAR, VARCHAR2, VIEW, WHENEVER, WHERE, WITH, ON, ONLINE, OPTION, OR, ORDER, PCTFREE, PRIOR, PRIVILEGES, PUBLIC, RAW, RENAME, RESOURCE, REVOKE, ROW, ROWID, ROWNUM, ROWS, SELECT, SESSION, SET, SHARE, SIZE, SMALLINT, START, SUCCESSFUL, SYNONYM, SYSDATE, TABLE, THEN, TO, TRIGGER, UID, UNION, UNIQUE, UPDATE, USER, VALIDATE	ABSOLUTE, ACTION, ADD, ADMIN, AFTER, AGGREGATE, ALIAS, ALLOCATE, ALTER, AND, ANY, ARE, ARRAY, AS, ASC, ASSERTION, AT, AUTHORIZATION, BEFORE, BEGIN, BINARY, BIT, BLOB, BOOLEAN, BOTH, BREADTH, BY, CALL, CASCADE, CASCADED, CASE, CAST, CATALOG, CHECK, CLASS, CLOB, CLOSE, COLLATE, COLLATION, COLUMN, COMMIT, COMPLETION, CONNECT, CONNECTION, CONSTRAINT, CONSTRAINTS, CONSTRUCTOR, CONTINUE, CORRESPONDING, CREATE, CROSS, CUBE, CURRENT, CURRENT_DATE, CURRENT_PATH, CURRENT_ROLE, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_USER, CURSOR, CYCLE, DATE, DAY, DEALLOCATE, DEC, DECIMAL, DECLARE, DEFAULT, DEFERRABLE, DEFERRED, DELETE, DEPTH, DEREF, DESC, DESCRIBE, DESCRIPTOR, DESTROY, DESTRUCTOR, DETERMINISTIC, DICTIONARY, DIAGNOSTICS, DISCONNECT, DISTINCT, DOMAIN, DOUBLE, DROP, DYNAMIC, EACH, ELSE, END, END-EXEC, EQUALS, ESCAPE, EVERY, EXCEPT, EXCEPTION, EXEC, EXECUTE, EXTERNAL, FALSE, FETCH, FIRST, FLOAT, FOR, FOREIGN, FOUND, FROM, FREE, FULL, FUNCTION, GENERAL, GET, GLOBAL, GO, GOTO, GRANT, GROUP, GROUPING, HAVING, HOST, IDENTITY, IGNORE, IMMEDIATE, IN, INDICATOR, INITIALIZE, INITIALLY, INNER, INOUT, INPUT, INT, INTEGER, INTERSECT, INTO, IS, ISOLATION, ITERATE, JOIN, KEY, LANGUAGE, LARGE, LAST, LATERAL, LEADING, LESS, LEVEL, LIKE, LIMIT, LOCAL, LOCALTIME, LOCALTIMESTAMP, LOCATOR, MAP, MATCH, MODIFIES, MODIFY, MODULE, NAMES, NATIONAL, NATURAL, NCHAR, NCLOB, NEXT, NO, NONE, NOT, NULL, NUMERIC, OBJECT, OF, OFF, ON, ONLY, OPEN, OPERATION, OPTION, OR, ORDINALITY, OUT, OUTER, OUTPUT, PAD, PARTIAL, PATH, POSTFIX, PREFIX, PREORDER, PREPARE, PRESERVE, PRIMARY, PRIOR, PRIVILEGES, PROCEDURE, READ, READS, REAL, RECURSIVE, REFERENCES, REFERENCING, RELATIVE, RESTRICT, RESULT, RETURN, RETURNS, REVOKE, ROLE, ROLLBACK, ROLLUP, ROUTINE, ROW, ROWS, SAVEPOINT, SCHEMA, SCROLL, SCOPE, SEARCH, SECTION, SELECT, SEQUENCE, SESSION, SESSION_USER, SET, SETS, SIZE, SMALLINT, SOME, SPECIFIC, SPECIFICTYPE, SQL, SQLEXCEPTION, SQLSTATE, SQLWARNING, START, STATIC, STRUCTURE, SYSTEM_USER, TABLE, TEMPORARY, TERMINATE, THAN, THEN, TIME, TIMESTAMP, TIMEZONE_HOUR, TIMEZONE_MINUTE, TO, TRAILING, TRANSACTION, TRANSLATION, TREAT, TRIGGER, TRUE, UNDER, UNION, UNKNOWN, UNNEST, UPDATE, USAGE, USING,VALUES, VARCHAR, VARIABLE, VARYING, VIEW, WHEN, WHENEVER, WHERE, WITH, WITHOUT, WORK, WRITE, ZONE

5.1.5 DATABASE OBJECT DESIGN CONCERNS

For Database Objects, or Schema Objects, users need to put many factors into account before migration. It will contain a variety of constraints checking, data types mapping and so on. We enlist the factors as followings.

- Data Types
- Entity Integrity Constraints
- Referential Integrity Constraints
- Unique Key Constraints
- Check Constraints

5.1.5.1 Entity Integrity Constraints

A primary key can be defined as part of a CREATE TABLE or an ALTER TABLE statement. Oracle internally creates a unique index to enforce the integrity.

So does DBMaster, a primary key constraint will be applied to a unique index internally. The performance will be promoted for it's an index too. The constraint will be kept to retainintegrity.

Oracle	DBMaster
CREATE TABLE table_name(Column1 datatype PRIMARY KEY, Column2 datatype, …); CREATE TABLE table_name(Column1 datatype, Column2 datatype, …, CONSTRAINT pk_name PRIMARY KEY (Column1, Column2,…));	CREATE TABLE table_name(Column1 datatype PRIMARY KEY, Column2 datatype, …); CREATE TABLE table_name(Column1 datatype, Column2 datatype, …, PRIMARY KEY (Column1, Column2,…));
ALTER TABLE table_name ADD PRIMARY KEY (column_name) ALTER TABLE Table_name ADD Constraint pk_name PRIMARY KEY (Column1, Column2,…);	ALTER TABLE table_name PRIMARY KEY (Column1, column2,…); ALTER TABLE table_name ADD CONSTRAINT pk_name PRIMARY KEY(Column1,column2,…);

5.1.5.2 Referential Integrity Constraints

Oracle provides declarative referential integrity. A CREATE TABLE or ALTER TABLE statement can add foreign keys to the table definition.

You can also define a foreign key for a table in DBMaster. Foreign keys can be defined in a CREATE TABLE statement or an ALTER TABLE statement.

DBMaster and Oracle have many similarities in the term of Integrity Constraints. It makes the migration process less labor.

Oracle	DBMaster
CREATE TABLE table_name1 (Column1 datatype NOT NULL PRIMARY KEY, Column2 datatype NOT NULL, Column3 datatype , FOREIGN KEY(column_name1) REFERENCES table_name2(column_name2)) CREATE TABLE table_name (Column1 datatype NOT NULL, Column2 datatype NOT NULL, Column1 datatype,… PRIMARY KEY (column_nameF), CONSTRAINT fk_name FOREIGN KEY (column_nameP) REFERENCES Persons(column_nameP))	CREATE TABLE table_name1(Column1 datatype, Column2 datatype, …, FOREIGN KEY fk_name(column1,…) REFERENCES table_name2); CREATE TABLE table_name(Column1 datatype, Column2 datatype, …, Column datatype CONSTRAINT fk_name REFERENCES table_name2(column_name));
ALTER TABLE table_name ADD FOREIGN KEY (column_name) REFERENCES Persons(column_name) ALTER TABLE table_name1 ADD CONSTRAINT fk_name FOREIGN KEY (column_name) REFERENCES table_name2(column_name)	ALTER TABLE table_name ADD FOREIGN KEY (column_name) REFERENCES Persons(column_name) ALTER TABLE table_name1 ADD CONSTRAINT fk_name FOREIGN KEY (column_name) REFERENCES table_name2(column_name) ALTER TABLE tb_name1 FOREIGN KEY(column1,column2,…) REFERENCES table_name2;

5.1.5.3 Unique key Constraints

Oracle defines unique keys as part of CREATE TABLE or ALTER TABLE statements. Oracle internally creates unique indexes to enforce these constraints.

You can also define a unique key for a table in DBMaster. Unique keys can be defined in a CREATE TABLE statement or an ALTER TABLE statement. However, in DBMaster, the unique key is referreded to as the the unique index. Users should be aware of the difference between the two terminologies.

Oracle	DBMaster
CREATE TABLE table_name1 (Column1 datatype NOT NULL **UNIQUE**, Column2 datatype, ,…) CREATE TABLE table_name1 (Column1 datatype NOT NULL, Column2 datatype, ,…, CONSTRAINT uc_name UNIQUE (colunm1, column2,…))	CREATE TABLE table_name1 (Column1 datatype NOT NULL **UNIQUE**, Column2 datatype, ,…) CREATE TABLE table_name1 (Column1 datatype NOT NULL, Column2 datatype, ,…, CONSTRAINT uc_name UNIQUE (colunm1, column2,…)) CREATE TABLE table_name1 (Column1 datatype CONSTRAINT u **UNIQUE**, Column2 datatype, ,…)
ALTER TABLE table_name ADD **UNIQUE** (column_name) ALTER TABLE table_name ADD CONSTRAINT uc_name **UNIQUE** (column1,column2,…)	ALTER TABLE table_name ADD **UNIQUE** (column_name) ALTER TABLE table_name ADD CONSTRAINT uc_name **UNIQUE** (column1,column2,…)

5.1.5.4 Check Constraints

Oracle defines check constraints as part of the CREATE TABLE statement or the ALTER TABLE statement. A check constraint is defined at the TABLE level and the COLUMN level.

Check constraints can be defined in a CREATE TABLE statement or an ALTER TABLE statement in DBMaster as well. Multiple check constraints can be defined on a table.

A table-level check constraint can refer to any column in the constrained table. A column can have only one check constraint. A column-level check constraint can refer to only the constrained column.

Table-level check constraints from Oracle databases map one-to-one with DBMaster check constraints. Furthermore, since DBMaster has the column-level check, migration from Oracle to DBMaster will have not lost the check constraints or sort of things.

Oracle	DBMaster
CREATE TABLE table_name(Column1 datatype CHECK (boolean_expression), Column2 datatype CHECK (check_expression), …); CREATE TABLE table_name(Column1 datatype , Column2 datatype,… CONSTRAINT ck_name CHECK (check_expression1 AND check_expression2 AND …));	CREATE TABLE table_name(Column1 datatype CHECK boolean_expression, Column2 datatype CHECK boolean_expression, …); CREATE TABLE table_name(Column1 datatype, Column2 datatype, [CONSTRAINT ck_name] CHECK(boolean_expression1 AND boolean_expression2 AND …) …);
ALTER TABLE table_name ADD CHECK (check_expression) ALTER TABLE table_name ADD CONSTRAINT ck_name CHECK (check_expression1 AND check_expression2,…)	ALTER TABLE table_name MODIFY (column1 to column1 datatype CHECK column1 boolean_expression,…);

5.2 Data Types Mapping

This section provides detailed descriptions of the differences in data types used by Oracle and DBMaster databases.

5.2.1 COMMON DATA TYPE MAPPING

Specifically, this section contains the following information:

- A table shows the base and available Oracle data types and how they are mapped to DBMaster data types.
- Recommendations based on the information are listed in the table:

Oracle	Description	DBMaster	Comments
Numeric Datatypes		Numeric Datatypes	
NUMBER(1-5)		SMALLINT	Two-byte integer, 15 bits, and a sign. (-2^15 – 2^15-1)
NUMBER(1-10)	The INTEGER data type is an exact signed numeric data type with a precision of 10 and a scale of 0.	INTEGER	The INTEGER data type uses 4 bytes of storage with the range of 2,147,483,647 to -2,147,483,648.

NUMBER(p, s)	Stored fixed and floating-point numbers. Its definition includes the precision and scale. The precision is between 1 and 38 and the scale can range from -84 to 127.	DECIMAL(p, s) NUMERIC(p, s)	The default value for precision is 17 with a maximum value of = 38. Scale refers to the number of digits to the right of the decimal point. The default value for scale is 6.
NUMERIC(p,s)	The precision can range from 1 to 38.	NUMERIC(p, s)	The maximum precision value is 38
DEC(p,s) DECIMAL(p,s)	Precision can range from 1 to 38 and the DECIMAL data type may be abbreviated as DEC.	DEC(p,s) DECIMAL(p, s)	The maximum precision value is 38 and the DECIMAL data type may be abbreviated as DEC.
FLOAT(1-24)	The FLOAT data type is an approximate signed numeric data type with a mantissa of precision 7. Precision refers to the total number of digits in the mantissa, both to the left and to the right of the decimal point.	FLOAT (REAL) (DB_FltDb=0)	The FLOAT data type uses 4 bytes of storage and has a valid input range of 3.402823466E38 to –3.402823466E38. The smallest valid input values are1.175494351E-38 and –1.175494351E-38.
FLOAT(1-53)	The DOUBLE data type is an approximate signed numeric data type with a mantissa of precision 15. Precision refers to the total number of digits in the mantissa, both to the left and to the right of the decimal point.	FLOAT(DOUBLE) (DB_FltDb = 1)	The DOUBLE data type uses 8 bytes of storage and has a valid input range of 1.0E308 to –1.0E308. For the data type FLOAT (54-) will need to truncate to the DOUBLE data type. Some precision will get lost for truncation.
Character datatypes		Character datatypes	
CHAR(1-2000)	The CHAR data type is a fixed-length data type that can contain any character from the keyboard. The CHAR length in Oracle is between 1 and 2000 bytes. Default is 1 byte. Padded on the right with blanks to full length of size.	CHAR (n byte) 3968 (4KB page size) 8064 (8KB page size) 16256 (16KB page size) 32640 (32KB page size)	In DBMaster, CHAR columns can be a minimum length of is 1 character and the maximum length Depending on DB_PGSIZ (4k, 8k, 16k, and 32k) (NO Unicode).

NCHAR(1-2000)	Maximum size of 2000 bytes. Fixed-length NLS string Space padded.	NCHAR (n byte) 1984 (4KB page size) 4032 (8KB page size) 8128 (16KB page size) 16320 (32KB page size)	The NCHAR data type is a fixed-length data type that can contain any Unicode character. NCHAR columns length can be Depending on DB_PGSIZ (4k, 8k, 16k, and 32k) ((Unicode).
VARCHAR2(1-4000)	The VARCHAR data type is a variable-length data type that can contain any character that can be entered from the keyboard. In Oracle, the length is between 1 and 4000 bytes.	VARCHAR 3968 (4KB page size) 8064 (8KB page size) 16**256 (16KB page size)** 32640 (32KB page size)	VARCHAR columns length can be Depending on DB_PGSIZ (4k, 8k, 16k, and 32k) (NO Unicode).
NVARCHER2(1-4000)	Maximum size of 4000 bytes. Variable-length NLS string.	NVARCHAR 1984 (4KB page size) 4032 (8KB page size) 8128 (16KB page size) 16320 (32KB page size)	The NVARCHAR data type is a variable-length data type that can contain any Unicode character. NVARCHAR columns length can be Depending on DB_PGSIZ (4k, 8k, 16k, and 32k) (Unicode).
Long Raw/BLOB	Raw binary data; otherwise the same as LONG. Oracle Long Raw can hold up to 2GB and the BLOB field can hold up to 4GB.	BLOB	DBMaster BLOB only holds up 8T.
RAW(1-2000)	The BINARY data type is a fixed-length data type that can contain any binary value. The length of RAW columns is between 1 byte and 2000 bytes.	BINARY(1-2000)	The minimum length of BINARY columns is 1 byte and the maximum length is 3992 bytes.
Large Object(LOB) Datatypes		Large Object(LOB) Datatypes	

BLOB	Stores unstructured binary large objects. The maximum length of BLOB is 4 GB.	LONG VARBINARY(BLOB)	The BLOB data type is a variable-length data type that can contain any binary value. The maximum length of BLOB columns is 8 TB.
CLOB	The CLOB data type is a variable-length data type that can contain any character that can be entered from the keyboard. The maximum length of CLOB is 4 GB.	LONG VARCHAR(CLOB)	The maximum length of CLOB columns is 8TB
LONG/CLOB	Oracle Long field can hold up to 2GB and CLOB field can hold up to 4 GB.	LONG VARCHAR(CLOB)	DBMaster CLOB holds up to 8TB.
NCLOB	Stores Unicode data and NCLOB field can hold up to 4 GB	N/A	
BFILE	Pointer to the external file Maximum file size of 2^{64}-1 bytes.	File	DBMaster provide the SYSTEM FO and User FO. Oracle BFILE is similar to DBMaster User FO.
Date/Time Datatypes		Date/Time Datatypes	
DATE	In Oracle, the date precision is to the second. A date between Jan 1, 4712 BC and Dec 31, 9999 AD.	TIMESTAMP	In DBMaster, the precision of TIMESTAMP is one second. The precision in Oracle is fitting to the TIMESTAMP in DBMaster
TIMESTAMP	In Oracle, the timestamp has a precision of 1/10000000th of a second.	TIMESTAMP	In Conversion, some précised data will be truncated when converting to TIMESTAMP data type in DBMaster
Rowid Datatypes		Rowid Datatypes	
ROWID	Fixed-length binary data. Every record in the database has a physical address or **rowid**	N/A	
UROWID(n)	Universal rowid. Where size is optional	N/A	

5.2.2 DATA TYPES MAPPING CONCERN

This section outlines conversion considerations for the Datetime and Image data type as examples to illustrate the factors you should consider:

- DATETIME Data Types
- IMAGE and TEXT Data Types (Binary/Character Large Objects)

5.2.2.1 DATETIME Data Types

The date/time definition and its precision in Oracle11g differ from the same name of data types in DBMaster. For example, Oracle also has a DATE data type that stores date and time values accurate to one second. But DATE data type only stores the date values in DBMaster. Another data type is TIMESTAMP in Oracle, which has a precision of 1/100000000th of a second. In DBMaster, TIMESTAMP has a precision of 1 second. According to the description here, Migration from Oracle to DBMaster would lose the precision of data in some cases.

Example:

```
Oracle:
CREATE TABLE example_table
(datetime_column date not null,
text_column long null,
varchar_column varchar2(10) null)
DBMaster:
CREATE TABLE example_table
(datetime_column timestamp not null,
text_column long varchar null,
varchar_column varchar(10) null)
```

5.2.2.2 BLOB/CLOB Data Types (IMAGE and TEXT Data Types)

The physical and logical storage methods for IMAGE and TEXT data in DBMaster differ from Oracle. Given the LONG VARCHAR and LONG VARBINARY data type, DBMaster will automatically allocate the physical storage. While the BLOB size is less than 3952 bytes（in 4k page size）, 8048bytes (in 8k page size),16240 bytes(in 16k page size),32624 bytes(in 32 k page size)the BLOB data could be stored together with normal data. If the data size is greater than 4K (4k page size for example), a pointer is used to indicate the LONG VARCHAR, LONG VARBINARY data. But the real BLOB data will be put into so-called ".BB" files. The other alternative is to use FILE data type. DBMaster uses FULL PATH link to indicate FILE data type. The physical data is stored externally as a file appearance.

This dynamical arrangement allows multiple columns of BLOB data per table and better performance. Similarity, in Oracle, big BINARY data may be stored in a BLOB type field and big CHAR data may be stored in a CLOB type field. Oracle also allows multiple BLOB and CLOB columns per table. BLOBS and CLOBS may or may not be stored in the row depending on their sizes.

After the version 4.0 of DBMaster, the keyword BLOB and CLOB are applied to LONG VARBINARY and LONG VARCHAR. In most cases, you don't have to rewrite the schema. But if the VARCHAR size is greater than 4K, you should use LONG VARCHAR instead of the original data type.

5.3 Index Mapping

A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of slower writes and increased storage space. Indexes can be created by using one or more columns of a database table, providing the basis for both rapid random look ups and efficient access of ordered records. The disk space required to store the index is typically less than that required by the table (since indexes usually contain only the key-fields according to which the table is to be arranged, and excludes all the other details in the table), yielding the possibility to store indexes in memory for a table whose data is too large to store in memory.

Oracle	Description	DBMaster	Comments
B*-tree indexes	B*-tree indexs is the standard type of indexs available in Oracle, and it's very useful for selecting rows that meet an equivalence criterion or a range criterion. In oracle, B*-tree indexs is regard as the default index mode, two-way list existed in the tree leaf nodes and it will be accelerate speed for index orientation, An Oracle database automatically creates a B-tree index for the primary key of a table and for columns included in a unique constraint. Indexes are created via the **create index** command. simplified syntax： CREATE [UNIQUE] INDEX index_name ON table_name(column_name[, column_name ...]) TABLESPACE tab_space;	**Index**	DBMaster supports as many as indexes per table, having no limit. But create an index on one or more columns, up to a maximum of 32 columns. DBMaster limits indexes to a maximum record size of 4000 bytes. Creating indexes for frequently used expressions will improve query performance. DBMaster creates a index by syntax: CREATE [UNIQUE] INDEX index-identifier ON base-table-name ({column-identifier | expression} [ASC|DESC]',...)[IN tablespace-name] [FILLFACTOR unsigned-integer]
Function-based indexes	Instead of indexing a column, such as Name, you can index a function-based column, such as **UPPER** (Name). The function-based index gives the Oracle optimizer additional options when selecting an execution path. For example: a function-based index CREATE INDEX idx_name ON table_name(UPPER(column_name)) ;	N/A	

Bitmap indexes	For columns that have few unique values, a bitmap index may be able to improve query performance. Bitmap indexes should only be used when the data is batch loaded (as in many data warehousing or reporting applications). Simply syntax: CREATE BITMAP INDEX idx_name ON table_name(column_name);	N/A	
Reverse key indexes	If there are I/O contention issues during the inserts of sequential values, Oracle can dynamically reverse the indexed values prior to storing them. Simply syntax: CREATE INDEX idx_anme ON table_name(column_name) REVERSE;	N/A	
Partitioned indexes	You can partition indexes to support partitioned tables or to simplify the index management. Index partitions can be local to table partitions or may globally apply to all rows in the table.	N/A	

Text indexes	You can index text values to support enhanced searching capabilities, such as expanding word stems or searching for phrases. Text indexes are sets of tables and indexes maintained by Oracle to support complex text-searching requirements. Oracle Database 11g offers enhancements to text indexes that simplify their administration and maintenance. TEXT INDEX mainly have two kinds CONTEXT and CTXCAT.simply syntaxes for them are: CREATE INDEX idx_name ON table_name(column_name) INDEXTYPE IS CTXSYS.CONTEXT; CREATE INDEX idx_name ON table_name(column_name) INDEXTYPE IS CTXSYS.CTXCAT;	**SIGNATURE TEXT INDEX**	Signature text indexes are built in the same tablespace as the column for which the index is being built. A text index provides fast access to rows that contain one or more words or phrases in columns containing text. Text indexes contain a representation of all the text found in the text columns they are based on. The data is encoded and structured to make retrieval much faster than directly from the table. Typically created on column by using Order By clause. Rebuild the text index if you load data after creating text indexs. Text index names must be unique for each table. Text index names have a maximum length of thirty-two characters, and may contain numbers, letters, the underscore character, and the symbols $ and #. The first character can not be a number. Create syntax: CREATE SIGNATURE TEXT INDEX text- index-identifier ON table_name(column_name,…) [TOTAL TEXT SIZE number] [MB SCALE number] [ORDER BY column_name ASC|DESC]

		IVF TEXT INDEX	IVF indexes are built in a separate file and exhibit better performance for larger indexes. An IVF text index can be used in place of a standard index to increase the performance of queries, particularly on columns that contain more than 200 MB data. IVF indexes are sorted in the operating system's file system, and are administered through the database. The location where the IVF index should be stored is specified when the index is created. DBMaster manages the creation of sub-directories within the IVF index root directory. Besides these special features, others are same as signatures of text indexs. Create syntax: CREATE IVF TEXT INDEX text-index-identifier ON table_name(column_name,…) [STORAGE PATH path] [TOTAL TEXT SIZE number MB] [ORDER BY column_name ASC|DESC]

5.4 Data Manipulation Language (DML)

This section uses tables to compare the syntax and description of Data Manipulation Language (DML) elements in Oracle and DBMaster. The following topics are present in this section:

- Connecting to the Database
- SELECT Statements
- SELECT with GROUP BY Statements
- INSERT Statements
- UPDATE Statements
- DELETE Statements
- Operators

- Comparison Operators
- Arithmetic Operators
- String Operators
- Set Operators
- Bit Operators
- Built-In Functions
- Character Functions
- Date Functions
- Mathematical Functions
- Locking Concepts and Data Concurrency Issues
- Locking
- Row-Level Versus Page-Level Locking
- Read Consistency
- Logical Transaction Handling

5.4.1 CONNECTING TO THE DATABASE

Oracle	DBMaster	Description
CONNECT user_name/password SET role	Connect to DB_NAME USER_NAME PASSWORD;	

Recommendations:

An Oracle Server controls only one database. In addition, in Oracle a user executes the SET ROLE command to change roles or re-issues a CONNECT command by using a different user_name. In DBMaster users is assigned to multiple groups or roles. It is only permitted to login as one role. Only the highest privilege will be activated after logging into the DBMaster Database System.

5.4.2 SELECT STATEMENTS

Oracle	DBMaster	Description
SELECT sysdate FROM dual;	SELECT curdate();	SELECT Statements without FROM Clauses
SELECT [ALL | DISTINCT] {select_list}	SELECT [ALL | DISTINCT] {select_list}	The ALL keyword means every record regardless of its duplicate occurrence. However, using DISTINCT keyword will eliminate the duplicate rows.

{select_list} [predicate]{* | [owner.]{table | view | snapshot | synonym}.* | {[owner.]table.column | constant_literal | expression } alias}}	{select_list} {* | [remote-table-name@][owner.]{table | view | synonym} | {[remote-table-name@][owner.]table.column | expression } [AS] alias}}	COLUMN ALIAS is defined by putting the alias directly after the selected COLUMN. This function is supported after DBMaster 4.0.3. You can also retrieve data from SYNONYMS. EXPRESSION could be a column name, a literal, a mathematical computation, a function, several functions combined, or one of several PSEUDO-COLUMNS.
SELECT … FROM [user.]{table | view } [@dblink] [alias] [, [user.] {table | view3} [@dblink] [ALIAS]...	SELECT … FROM [[db-name[@server-name]:]user.]|[databaselink]table-name|view] [ALIAS]	The main difference is DBMaster must use DB_NAME@HOST:USERNAME. TABLE_NAME as its full valid table name. It would take effort to transfer the Oracle @dblink to its corresponding name in DBMaster. DBMaster also supports database link currently.
SELECT …FROM …[WHERE] [GROUP BY (column-identifier [, column-identifier]…)] [HAVING search-condition] [ORDER BY sort-specification] [WITH OWNERACCESS OPTION]	SELECT …FROM …[WHERE] [GROUP BY (column-identifier [, column-identifier]…)] [HAVING search-condition] [ORDER BY sort-specification]	Aggregate functions and syntax are identical in both databases. If a GROUP BY clause is used, all non-aggregate select columns must be in a GROUP BY clause.
INSERT INTO <table> SELECT FROM....	INSERT INTO <table> SELECT FROM.... SELECT FROM.. INTO <table>	You could retain the original syntax in Oracle or rewrite it to the 2nd syntax. It allows you to insert the results of the SELECT statement into a table.
SELECT … FROM (SELECT…) ALIAS	SELECT …FROM … WHERE…(SELECT…) ALIAS SELECT … into tmp_table SELECT … FROM tmp_table	Sub-queries in Place of Columns are supported in Oracle. DBMaster is also support that after version 4.1. Before that, the users need to use temp table to do the conversion

SELECT * from tab1, tab2 WHERE tab1.col1 = tab2.col1 (+); SELECT * from tab1 RIGHT OUTER JOIN tab2 on (tab1.col1=tab2.col1) SELECT * FROM tab1 RIGHT OUTER JOIN tab2 USING(col1)	SELECT * from tab2 outer join tab1 where tab1.col1=tab2.col1; SELECT * from tab1 RIGHT OUTER JOIN tab2 on (tab1.col1=tab2.col1) SELECT * FROM tab1 RIGHT OUTER JOIN tab2 USING(col1)	For Oracle8i or previous version, the outer join must use + sign to indicate which is the based table. As with Oracle8i, DBMaster 3.7X only support “outer join” syntax. But in both DBMaster 5.1 and Oracle 11g, the ANSI-compliant syntax all works. The user should rewrite the syntax to ANSI standard for compatibility concern.
SELECT …FOR UPDATE	SELECT …FOR UPDATE	Both Oracle and DBMaster could use this syntax to lock the table. Note that without “FOR UPDATE”, both of them won’t lock any tables.

Recommendations:

For SQL-99 , most of SQL commands are compatible either in Oracle or in DBMaster. For “SELECT” command, Oracle uses read-committed transaction level yet DBMaster assumes read-uncommitted level. This difference will have users to check the flaw and logics of Oracle application discreetly. In most cases, this difference won’t lead into any troubles. But it might cause some inconsistency or integrity problems given special case. Users could use “FOR BROWSE” to have a share lock on some specific tables and make it consistent with “read-committed” level.

5.4.3 INSERT STATEMENTS

Oracle	DBMaster	Description
INSERT INTO [user.]{table | view}[@dblink][(column [, column]...)]{VALUES (expression [, expression]...) | query...};	INSERT INTO remote-table-name [(column-identifier [, column-identifier]...)] { VALUES (insert-value[,insert-value]...) | DEFAULT VALUES | select-order-by-statement }	Insertings can only be done on single table views.

Recommendations:

The values supplied in the VALUES clause in either database may contain functions. The Oracle-specific functions must be replaced with the equivalent DBMaster ones.

5.4.4 UPDATE STATEMENTS

Oracle	DBMaster	Description:
UPDATE [user.]{table | view} [@dblink] SET [[user.] {table. | view.}] { column = expression | NULL | (select_statement) [, column = expression | NULL | (select_statement)...] | (column [, column]...) = (select_statement)} [WHERE {condition | CURRENT OF cursor}]	UPDATE [remote-table-name@][owner.]{table | view } [table-option] SET column-identifier = {expression | subquery | NULL} [, column-identifier = {expression | subquery | NULL}]... [WHERE CURRENT OF cursor-name] | [WHERE search-condition]	A single subquery may be used to update a set of columns. This subquery must select the same number of columns (with compatible data types) as are used in the list of columns in the SET clause. The CURRENT OF cursor clause causes the UPDATE statement to affect only the single row currently in the cursor as a result of the last FETCH. The cursor SELECT statement must have included in the FOR UPDATE clause. Updates can only be done on single table views.

5.4.5 DELETE STATEMENTS

Oracle	DBMaster	Description:
DELETE [FROM] [user.]{table | view}[@dblink] [alias] [WHERE where_clause]	DELETE FROM remote-table-name [table-option] [WHERE search-condition]	FROM is optional in Oracle but elementary in DBMaster. Deletings can only be performed through single table views in both Oracle and DBMaster

5.4.6 OPERATORS

5.4.6.1 Operator comparison

Operator	Same in both Databases	Oracle only	DBMaster only
Equal to	=		
Not equal to	!=, <>	^=	
Less than	<		
Greater than	>		
Less than or equal to	<=		
Greater than or equal to	>=		
Greater than or equal to x	BETWEEN x AND		

and less than or equal to y	y		
Pattern Matches	LIKE 'a\%' ESCAPE '\'		Contain, Match
a followed by 0 or more characters	LIKE 'a_'		
Does not match pattern	NOT LIKE		
No value exists	IS NULL		
A value exists	IS NOT NULL		
At least one row returned by query	EXISTS (query)		
No rows returned by query	NOT EXISTS (query)		
Equal to a member of set	IN, =ANY	=SOME	
Not equal to a member of set	NOT IN,!= ANY, <> ANY	!= SOME <> SOME	
Less than a member of set	< ANY	< SOME	
Greater than a member of set	> ANY	> SOME	
Less than or equal to a member of set	<= ANY	<= SOME	
Greater than or equal to a member of set	>= ANY	>= SOME	
Equal to every member of set	=ALL		
Not equal to every member of set	!= ALL, <> ALL		
Less than every member of set	< ALL		
Greater than every member of set	> ALL		
Less than or equal to every member of set	<= ALL		
Greater than or equal to every member of set	>= ALL		
Add	+		
Subtract	-		
Multiply	*		
Divide	/		
Modulo	Mod(x,y)		

Concatenate	||, Concat(substr1,substr2)		
Identify Literal	'this is a string'		
Distinct row from either query	UNION		
All rows from both queries	UNION ALL		
All distinct rows in both queries		INTERSECT	
All distinct rows in the first query but not in the second query		MINUS	

5.4.6.2 Search String methods

DBMaster supports "MATCH", "CONTAIN", "CONTAINS", and "LIKE" for pattern search.

Basically, "Like" operators will scan the whole record and seek the pattern as a token. DBMaster provides another operator "CONTAIN" to seek the word fragment. In addition, users could use "MATCH" operator to seek the full word. Only the MATCH and CONTAINS operators can be applied to a text index search.

Oracle supports LIKE operator in a WHERE clause to search a string for a pattern. You can specify patterns using a combination of normal characters and the following two wildcard characters:

- Underscore (_) Matches one character in a specified position
- Percent (%) Matches any number of characters beginning at the specified position

Oracle	**Usages**	**DBMaster**	**Usages**
LIKE	The LIKE condition allows you to use wildcards in the **where** clause of an SQL statement. This allows you to perform pattern matching. The LIKE condition can be used in any valid SQL statement - select, insert, update, or delete. The patterns that you can choose from are: % allows you to match any string of any length (including zero length) _ allows you to match on a single character	LIKE	This takes the form: x LIKE 'y' ESCAPE 'z'; the LIKE condition is satisfied when the string value or expression to the left of the LIKE keyword meets the criteria specified in the case-sensitive quoted string to the right of the keyword. Syntax: Expression [NOT] LIKE Condition
N/A		contains	The contains operator's condition is satisfied when the concatenated string from concatenate columns

			matches the string pattern. Used in Full-Text-Index, so Create TEXT-INDEX first before using CONTAINS predicate. Syntax: CONTAINS (column_name,' contains_search_condition ');
N/A		match	This takes the form: x NOT CASE MATCH ‘y’; the MATCH condition is satisfied when the quoted string to the right of the MATCH keyword matches the entire string value or expression to the left of the keyword. The NOT keyword inverts the search results and CASE keywords keyword makes the search case-sensitive, Syntax: Column_name MATCH Condition
N/A		contain	This takes the form x NOT CASE CONTAIN ‘y’; the CONTAIN condition is satisfied when the quoted string to the right of the CONTAIN keyword matches any part of the string value or expression to the left of the keyword. The NOT keyword inverts the search results and the CASE keyword makes the search case-sensitive, both are optional Syntax: column_name CONTAIN ‘condition’;

Users must know how to translate the corresponding functions in DBMaster.

For example：

Table: Dept

ID	**Name**
1	DBMaster Support
2	Support SQL Server
3	SQL Server DBMaster

SQL1:

```
Oracle：
SQL>  select * from Dept where Name like 'DBMaster S_';
no rows selected
DBMaster：
dmSQL>  select * from Dept where Name like 'DBMaster S_';
    ID                            NAME
    ========= ======================================
0 rows selected
```

SQL2:

```
Oracle:
SQL> select * from Dept where Name like 'DBMaster S%';
        ID NAME
---------- ----------------------------------------
 1 DBMaster Support
DBMaster:
dmSQL> select * from Dept where Name like 'DBMaster S%';
ID                            NAME
====== ==================
    1 DBMaster Support
1 rows selected
```

SQL3:

```
Oracle
SQL> select * from Dept where Name like '%DBM%';
        ID NAME
---------- --------------------------------------------------
         1 DBMaster Support
         3 Oracle DBMaster
DBMaster:
dmSQL> select * from Dept where Name contain 'DBM';
    ID                            NAME
========== ================================================
          1 DBMaster Support
          3 Oracle DBMaster
2 rows selected
```

SQL4:

```
Oracle:
SQL> select * from Dept where Name like '%DBMaster%';
        ID NAME
---------- --------------------------------------------------
         1 DBMaster Support
         3 Oracle DBMaster
DBMaster:
dmSQL> select * from Dept where Name match 'DBMaster';

    ID                            NAME
========== ================================================
          1 DBMaster Support
          3 Oracle DBMaster
2 rows selected
```

SQL5:

```
Oracle:
SQL>  select * from Dept where Name like 'DBMaster S' ;
no rows selected
DBMaster:
dmSQL> select * from Dept where Name match 'DBMaster S' ;
    ID                            NAME
========== ================================================


0 rows selected
```

5.4.6.3 Special operators recommendation

- **Convert the "SOME" operator to its logical correspondent**

DBMaster doesn't support "SOME" operator, on the other hand, most of "SOME" embedded SQL command could be replaced with another statement, remaining the same logical. For example, if "=SOME" was encountered, the user could easily replace the "=SOME" with "IN" or "=ANY".

- **No "Intersect" and "Minus" operators supported**

Oracle provides three methods to handle set of results, i.e., UNION, INTERSECT and MINUS. Currently, DBMaster only supports "UNION" operators. As to the "Intersect" and "Minus", users need to rewrite applications to do the further handling.

- **DBMaster supports "Contain" and "Match" for pattern search**

Basically, "Like" operator will scan the whole record and seek the pattern as a token. DBMaster provides another operator "CONTAIN" to seek the word fragment. In addition, users could use "MATCH" operator to seek the full word.

5.4.7 BUILT-IN FUNCTIONS

The user who read the following table and functions listed will get surprise that Oracle had so many common functions as DBMaster. We classify all the functions into four categories:

- Math/Number Functions
- Character Functions
- Conversion Functions
- Date Functions

It doesn't include all of Oracle functions. For example, Oracle has some Object-Reference functions, such as REF, DEREF. This kind of functions is rarely seen in any other RDBMS. DBMaster, as a pure RDBMS, can't implement such functions. In addition, some unique functions to Oracle, such as CHARTOROWID, NumToDSInterval and so on have not been put here for its uniqueness.

In most cases, users would need very little effort to migrate Oracle functions to DBMaster functions, the Oracle unique functions or Object-Reference functions are not commonly seen after all.

5.4.7.1 Math/Number Functions:

The numeric functions to perform calculations. These functions accept an input number, this may come from a numeric column or any expression that evaluates to a number. A calculation is then performed and a number is returned.

Oracle	DBMaster	Description
ABS(n)	ABS(n)	Return the absolute value of x as a double-precision floating-point number.
ACOS(n)	ACOS(n)	Return the arc cosine of n in the range 0 to pi as a double-precision floating-point number.
ASIN(n)	ASIN(n)	Return the arcsine of n in the range -pi/2 to pi/2 as a double-precision floating-point number.
ATAN(n)	ATAN(n)	Return the arc tangent of n in the range -pi/2 to pi/2 as a double-precision floating-point number.
ATAN2(x,y)	ATAN2(x,y)	Return the arc tangent of x/y in the range -pi to pi as a double precision floating-point number.
BITAND(x,y)	N/A	Return the result of performing a bitwise AND on *x* and *y*.
CEIL(n)	CEILING(n)	Return the least integral value greater than or equal to n as a double-precision floating-point number.
COS(n)	COS(n)	Return the cosine of n as a double-precision floating-point number. n is expressed in radians.
COSH(n)	COSH(n)	Return the hyperbolic cosine of *x*.
EXP(n)	EXP(n)	Return the exponential function e**x as a double-precision floating-point number.
FLOOR(n)	FLOOR(n)	Return the greatest integral value less than or equal to x as a double-precision floating-point number.
LN(n)	LOG(n)	Return the natural logarithm of x as a double-precision floating-point number.
LOG(base,number)	LOG10(n)	Return the logarithm to base 10 as a double-precision floating-point number.
MOD(m,n)	MOD(m,n)	Return the remainder (modulus) of m divided by n as a double-precision floating-point number.

POWER(m,n)	POW(m,n) POWER(m,n)	Returns x**y as a double-precision floating-point number.
ROUND(n[,m])	ROUND(n[,m])	Return the closest integer number of the real number x.
SIGN(n)	SIGN(n)	Return the sign of a number codes as +1 for positive, 0 for zero, and -1 for negative. Returns an integer value 1, 0 or -1.
SIN(n)	SIN(n)	Return the sine of n as a double-precision floating-point number. n is expressed in radians.
SINH(n)	SINH(n)	Return the hyperbolic sine of *x*
SQRT(n)	SQRT(n)	Return a double-precision floating-point number y where x = y*y.
TAN(n)	TAN(n)	Returns the tangent of n as a double-precision floating-point number. n is expressed in radians.
TANH(x)	TANH(x)	Return the hyperbolic tangent of *x*.
TRUNC(x[,y])	N/A	Return the result of truncating *x* to an optional y decimal places. If y is omitted, x is truncated to zero decimal places. If y is negative, x is truncated to the left of the decimal point.
N/A	DEGREES(n)	Return the number of degrees in radians as a double precision floating-point number
N/A	RADIANS(n)	Return the number of radians in degrees as a double precision floating-point number.
N/A	PI()	Returns the constant value of p, 3.1415926535897936, as a decimal number with a precision of 38 and a scale of 16.
N/A	RAND ()	Return a random Integer value.

5.4.7.2 Character Functions:

Character functions accept character input, which may come from a column in a table or, more generally, from any expression. This input is processed and a result is returned.

Oracle	DBMaster	Description
ASCII(char)	ASCII(string)	Return the ASCII code value of the leftmost character of string_exp as an integer. These 2 functions between DBMaster and Oracle are identical.
CHR(integer_expression)	CHAR(INT code)	Convert the decimal code for an ASCII character to the corresponding character. These 2 functions between DBMaster and Oracle are identical.
INSTR(STRING string_exp, STRING substring, INT start, INT occurrence)	LOCATE(STRING string_exp1, STRING string_exp2, INT start)	Return the starting position of the first occurrence of string_exp1 within string_exp2, The search for the first occurrence of string_exp1 begins with the first character position in string_exp2 unless the optional argument, start, is specified. (1) If either string_exp1 or string_exp2 is null, the result should be null (2) If start is null, return 0 (3) If string_exp1 is an empty string, the result should be 1 Oracle INSTR function will find the substring within string_exp from start position until the occurrence was found. It will need to do a little coding if the complex parameter was provided.
LENGTH(string) VSIZE(string)	LENGTH(string) CHAR_LENTTH(string) CHARACTER_LENTTH(string)	Compute the length allocated to an expression, giving the result in bytes. These 2 functions between DBMaster and Oracle are identical.
SUBSTR(char_exp, int start, int length)	SUBSTRING(string_exp, int start, int length)	Return the part of the string. Note that starting position in DBMaster must be negative. However, Oracle could use negative to indicate scaning the string backward.

COALESCE (expr1, expr2, ... expr_n)	COALESCE (expr1, expr2, ... expr_n)	In Oracle/PLSQL, the **COALESCE** function returns the first non-null expression in the list. If all expressions evaluate to null, then the coalesce function will return null. The coalesce function will compare each value, one by one. This function between DBMaster and Oracle are identical. COALESCE function is equivalent to the IF-THEN-ELSE statement For example: COALESCE (expr1, expr2, expr3… expr n) is equivalent to “if expr1 IS NOT NULL then expr1 else if expr2 IS NOT NULL then expr3 else….else expr _n”,In Oracle, you only need replace “else if” with “ELSIF”
NVL(variable, new_value)	COALESCE (variable, new_value)	If the value of the variable is NULL, the new_value is returned.
NVL2 (variable, *value1*, *value2*)	N/A	Return value1 if variable is not null; otherwise value2 is returned.
DECODE(exp,search1, result1, search2, result2, default)	1. CASE input_pression WHEN when_expression THEN result_expressuion[…., n][ELSE else_result_expression] END 2.CASE WHEN exp1 THEN result1 WHEN exp2 THEN result2 ELSE default_value END	DECODE compares expr to each search value one by one. If expr is equal to a search, then returns the corresponding result. If no match is found, then Oracle returns default. If default is omitted, then Oracle returns null. In DBMaster, you can use CASE WHEN…THEN… WHEN…THEN… END to implement the same function.
RPAD(char_exp, LENGTH(char_exp)*n, ' ')	REPEAT(string_exp, int count)	Produces a string with char_ exp repeated n times.
LPAD(char_exp, width [, pad_string])	N/A	In oracle, pad char_xep with spaces to the left to bring the total length of the string up to width characters.
UPPER(char_exp)	UPPER(String), UCASE(String)	Convert lowercase characters to uppercase characters. These three functions between DBMaster and Oracle are identical.
LOWER(char_exp)	LOWER(string), LCASE(string)	Convert all upper case characters in string_exp to lower case. These three functions between DBMaster and Oracle are identical.

LTRIM(char_exp)	LTRIM(char_exp)	Truncate trailing spaces from the left end of char_exp. These two functions between DBMaster and Oracle are identical.
RTRIM(char_exp)	RTRIM(char_exp)	Truncate the trailing spaces from the right end of char_exp. These two functions between DBMaster and Oracle are identical.
TRIM(char_exp)	TRIM(char_exp)	Truncate the trailing spaces from both end of char_exp. These two functions between DBMaster and Oracle are identical.
REPLACE(char,search_string,replacement_string)	REPLACE(string_exp1, string_exp2, string_exp3)	Replace all occurrences of string_exp2 in string_exp1 with string_exp3. These two functions between DBMaster and Oracle are identical.
SOUNDEX(string_exp)	N/A	Return the numeric difference of the SOUNDEX values of the string. DBMaster hasn't supported this yet.
N/A	RIGHT(string_exp1,n)	Return the rightmost count characters in string
N/A	LEFT(string_exp1,n)	Return the leftmost count characters in string
CONCAT(char1,char2) ||	CONCAT(string_exp1,string_exp2) ||	Return a character string that is the result of concatenating string_expr2 to string_ep1. The resulting string is DBMS dependent. In DBMaster 4.3, we use the same operand (||) as Oracle, users can save the work of converting "||" to "concat".
INITCAP(*x*)	N/A	Convert the initial letter of each word in x to uppercase and return the new string.

5.4.7.3 Conversion Functions:

Sometimes you need to convert a value from one data type to another. For this purpose, you should use a conversion function.

Oracle	**DBMaster**	**Description**
CAST(Column as Datatype)	CAST(Column as Datatype)	The **cast** function allows the output data to be converted to another data type. This function between DBMaster and Oracle is identical.
TO_CHAR(Column [,FORMAT Model])	CAST(Column as Datatype)	DBMaster uses CAST function to cast one data type to another. Oracle is able to set up some format model to format the output string. i.e., TO_CHAR (100) =CAST (100 as char (3)). However, if the complex format model is used, users will need to do a little big coding by himself.

TO_NUMBER(Column [,FORMAT Model])	STRTOINT() CAST(Column as Datatype)	The STRTOINT function converts the string to an integer, when the string argument is NULL, a NULL value is returned. An error is returned if the string cannot be converted to an integer. i.e., TO_NUMBER ('100') =CAST ('100' as INT). However, if the complex format model is used, users will need to do a little big coding by himself.
TO_DATE(Column, [,FORMAT Model])	TO_DATE(Column, [,FORMAT Model])	The TO_DATE function converts a selected string to a DATE format. This function between DBMaster and Oracle is identical. But in DBMaster before using it you should create it first manually by following command: create function to_date.TO_DATE(varchar(20), varchar(20)) RETURNS DATE;
RAWTOHEX(Column)	N/A	In DBMaster, the binary datatype would be output as string type automatically. Therefore, it has no needs to do extra conversion.
HEXTORAW(STRING string_exp)	'string_exp'x	In DBMaster, any string followed by 'x' character indicates the string should be treated as Hex Number. For example, HEXTORAW('7D')='7D'x

5.4.7.4 Date Functions:

Oracle	DBMaster	Description
date+int_exp requires conversion of int_exp to a number of days	ADD_DAYS(DATE date_val,INT s)	Add the int_exp number of days to the date contained in datetime_var.
date2-date1	DAYS_BETWEEN(DATE date1,DATE date2)	Return the number of days between the given two dates. date1 can be earlier or later than date2.
ADD_MONTHS (date, int_exp)	ADD_MONTHS(DATE date_val,INT s)	Return a date which is got from adding s months to date_val. s can be a negative number.
CURRENT_DATE SYSDATE	CURDATE()	Return current date.
CURRENT_TIMESTAMP	NOW()	Return current date and time as a timestamp value.

EXTRACT(portion from date_val) TO_CHAR(date, format)	YEAR(date),MONTH(date),WEEK(date), QUARTER(date), DAYNAME(date), DAYOFYEAR(date), DAYOFMONTH(date), DAYOFWEEK(date), DATEPART(date), TIMEPART(date), MDY(date), HMS(date), HOUR(date), MINUTE(date), SECOND(date)	Return the specified part of the date as an integer.
LAST_DAY(dateval)	LAST_DAY(dateval)	Return the last date of the month which dateval belongs to.
MONTHS_BETWEEN (date2, date1)	DAYS_BETWEEN(date1,date2)/30	Return the difference between the dates specified by the datetime1 and datetime2 variables. This difference is calculated in the number of months.
NEXT_DAY(dateval,datechar)	NEXT_DAY(dateval,weekday)	Return the date of the next first WeekDay.
SYSDATE	NOW()	Return the system date.

5.4.8 LOCKING CONCEPTS AND DATA CONCURRENCY ISSUES

Oracle	**DBMaster**
➢ Oracle supports table-lock and row-lock. ➢ Oracle supports the “Read Committed”,”Serializable”, “Read with shared lock”.	➢ DBMaster supports table-lock, page lock and row-lock. ➢ DBMaster supports the “Dirty Read”, “Read with shared lock”, and “Read with exclusive lock”.

Recommendations:

In Oracle, however, SELECT statements do not block UPDATE statements; this is a so-called READ-COMMITED isolation level. DBMaster, in the other way, uses the READ-UNCOMMITED as its isolation level. You should use the “select * for update” to prevent other session from updating the locked data. Basically, the reader of the data is never blocked both in Oracle and DBMaster. But users should be aware of the different manners when a “select” command is submitted on these two databases and their consequences.

Oracle	DBMaster
➢ Oracle transactions are implicit. ➢ Statements are not automatically committed to the database. The COMMIT WORK statement is required to commit the pending changes to the database. ➢ COMMIT WORK commits the pending changes to the database. ➢ ROLLBACK undoes all the transactions after the last COMMIT WORK statement. ➢ Savepoints can be set in transactions with the following command: ➢ SET SAVEPOINT savepoint_name ➢ The following command rolls back to the specified SAVEPOINT; ➢ ROLLBACK <savepoint_name> ➢ Two-phase commit is automatic and transparent in Oracle. Two-phase commit operations are needed only for transactions, which modify data on two or more databases.	➢ DBMaster transactions are explicit. ➢ Statements are automatically committed to the database by default. But user could change the DB_ATCMT=0 to change this pattern, or use “set autocommit off” to achieve the same effect. ➢ COMMIT WORK commits the pending changes to the database. ➢ ROLLBACK undoes all the transactions after the last COMMIT WORK statement. ➢ Savepoints can be set in transactions with the following command: ➢ SET SAVEPOINT savepoint_name ➢ The following command rolls back to the specified SAVEPOINT; ➢ ROLLBACK <savepoint_name> ➢ Two-phase commit is automatic and transparent in DBMaster. Two-phase commit operations are needed only for transactions, which modify data on two or more databases.

Recommendations:

Transactions are not implicit in DBMaster. Therefore, applications expect that every statement they issue is automatically committed after it is executed. But you could use “DB_ATCMT=1” in dmconfig.ini to change this manner.

Oracle transactions are always implicit, which means that individual statements are not committed automatically. When converting an Oracle application to a DBMaster application, care needs to be taken of determining what constitutes a transaction in that application. In general, a COMMIT work statement needs to be issued after every "batch" of statements, single statement, or stored procedures.

In DBMaster, transactions may also be explicitly begun by a client application by issuing a BEGIN TRAN statement during the conversion process.

Unlike Oracle, AUTOCOMMIT is default manner in DBMaster. Thus DBMaster will automatically commit every statement and write the change to Journal Files. This difference sometimes will make users have an impression that performance in DBMaster is inferior to Oracle .As a matter of fact, too many I/O for writing journals delays the operations. Users could easily find greatly improvement if turning off the “AUTOCOMMIT” option.

5.4.9 UDF DIFFERENCE

A UDF is a method that can be called in the context of a statement, can take any number of parameters, and can return any type of data.

Oracle	DBMaster
CREATE [OR REPLACE] FUNCTION function_name [(parameter_name [IN | OUT | IN OUT] type [, ...])] RETURN type {IS | AS} BEGIN function_body END function_name;	CREATE FUNCTION <udf_dll_name.function_name> (<function_datatype>) RETURN <function_output_datatype>;
SELECT function_name(parameter) FROM dual;	SELECT<function_name> (<related_table_column_name>) FROM <related_table>;
DROP FUNCTION function_name;	DROP FUNCTION <function_name>;

Recommendations:

Both Oracle and DBMaster allow programmers to build their own user-defined functions (UDF). Once a UDF has been written in Oracle or DBMaster, it is treated as a new built-in function with the same usages. You call your own functions as you would call any of the built-in database functions.

There is great difference exist between Oracle and DBMaster although both of them have UDF objects.

In Oracle, we can create user defined function (UDF) by PL/SQL script language so it with more powerful in Access to the database. Whereas DBMaster use C language as the carrier to create UDF, More embodies the function and feature of C language. But it doesn't represent very well in data access aspect, Because of this user need to spend time and do a careful technical evaluation if using UDF in your application programs.

5.4.10 TRIGGER DIFFERENCE

syntax level	
Oracle	**DBMaster**
CREATE [OR REPLACE] TRIGGER trigger_name {BEFORE | AFTER | INSTEAD OF | FOR} trigger_event ON table_name [REFERENCING [NEW AS <new_row_name>] [OLD AS <old_row_name>]] [FOR EACH ROW[WHEN (<trigger_condition>)]] [{FORWARD | REVERSE} CROSSEDITION] [{FOLLOWS | PRECEDES} schema.other_trigger}] [{ENABLE | DISABLE}] [WHEN trigger_condition]] BEGIN trigger_body END trigger_name;	Create Trigger trigger_name {Before|After} {Insert|Delete|Update[OF column_name]} On Table_name {FOR EACH ROW|FOR EACH STATEMENT}[When trigger_condition] trigger_body
ALTER TRIGGER trigger_name {disable|enable}	ALTER TRIGGER trigger_name REPLACE WITH
Drop trigger ledger_def_upd_row	DROP TRIGGER Trigger_name FROM Table_name
Trigger events	

There are three trigger events in the Oracle： BEFORE AFTER INSTEAD OF FOR(new for Oracle Database 11g)	There are four trigger events in the DBMaster ： BEFORE…FOR EACH STATEMENT”,”BEFORE…FOR EACH ROW”,”AFTER…FOR EACH STATEMENT” ,”AFTER…FOR EACH ROW”
function support	
Reference objects： Oracle support Trigger applies to both **table** and **view** object	**Reference objects：** DBMaster only support **Table** object currently.
action time： BEFORE,AFTER	**action time：** BEFORE, AFTER
trigger type： Row-level AND statement-level triggers	**trigger type：** Row triggers and statement triggers.
data reference： NEW and OLD keywords hold the new values or old values of the rows that may be changed by the user action in Oracle	**data reference：** DBMaster has NEW and OLD buffer corresponding with new and old in Oracle.

Recommendations:

There are lots of differences between Oracle and DBMaster in the trigger. The simplified description for the difference is as follows:

- Syntax level

In Oracle, most DDL statements to create schema object have the “Or Replace” to replace the original schema object if it exists. However, DBMaster doesn’t provide such syntax when creating the schema object. Users must remove the original “Or Replace” from the Oracle syntax.

Oracle supports more syntax than DBMaster for Trigger, you will find it from above creating syntax. Users must remove the original OPTIONS which supported by Oracle but not by DBMaster from Oracle syntax when migration.e.g “FORWARD”,” REVERSE”,” CROSSEDITION”, etc. More details please refer to the **Oracle On-Line Help** if interested.

- Trigger event

There are four trigger events in Oracle, including the”BEFORE”,”AFTER”,”INSTEAD OF”,”FOR” and four trigger events in the DBMaster including the“ BEFORE…FOR EACH STATEMENT”,”BEFORE…FOR EACH ROW”,”AFTER…FOR EACH STATEMENT” ,”AFTER…FOR EACH ROW”.

In Oracle, FOR EACH ROW means the trigger is a row-level trigger, that is, the code contained within trigger_body runs for each row when the trigger fires. If you omit FOR EACH ROW, the trigger is a statement-level trigger, that means the code within trigger_body runs once when the trigger fires. But INSTEAD OF trigger statements are implicitly activated for each row.

In the DBMaster, the keyword “FOR EACH ROW” means that if one record was modified in the database then the trigger would be executed. The keyword” FOR EACH STATEMENT” means executing the action before or after the SQL statement. It is similar with Oracle.

One of the most powerful features of a trigger is the ability to use a stored procedure as a trigger action both in DBMaster. Oracle also can call a procedure in trigger body. Therefore, if some complex job cannot be done in DBMaster, try to implement it in triggering Stored Procedures.

Users can't insert into multiple table by a single trigger action both in Oracle and DBMaster. Thus, users has to program a stored procedure to see that single action triggers to insert multiple tables. Despite that, the syntax between these two databases is similar.

- function support

Firstly, Reference objects, in Oracle, you can create only BEFORE and AFTER triggers for tables. INSTEAD of triggers are only available for views, typically they are used to implement view updates.

For DBMaster, only supporst Table objects currently.

Second, about trigger action time, Trigger for INSERT, UPDATE, and DELETE operation can be specified on a table or a view. Both Oracle and DBMaster support BEFORE and AFTER firing.

Third, about trigger type, Both Oracle and DBMaster support row triggers and statement triggers. Namely either row or table data once change trigger will be fire.

Last one is about trigger data reference DML triggers using the, NEW and OLD keywords hold the old values or new values of the rows that may be changed by the user action in Oracle. DBMaster is identical with Oracle.

Worthy of noting, In Oracle, about NEW and OLD usages have many restrictions, the special variables NEW and OLD are available to refer to new and old tuples respectively, used only for row-level triggers. Namely, the trigger which use these two keywords must include FOR EACH ROW statement,: NEW and: OLD variables are array variable which provide by system automatically: NEW used to record newly inserted data and OLD used to record the deleted. More detailed information, please refer to the **Oracle On-Line Help** if interested.

5.4.11 STORED PROCEDURES AND STORED FUNCTIONS

Oracle	**DBMaster**
CREATE [OR REPLACE] PROCEDURE procedure_name [(parameter_name [IN | OUT | IN OUT] type [, ...])] {IS | AS} [local_var data type;]… BEGIN procedure_body END procedure_name;	CREATE PROCEDURE procedure-name [(procedure-parameter [, procedure-parameter ...])] { [RETURNS STATUS] | [RETURNS [STATUS,] procedure-result [,procedure-result ...]] } | CREATE PROCEDURE FROM source-file-path
EXECUTE sp_name [(parameter,…)]	CALL sp_name [(parameter,…)];

ALTER PROCEDURE procedure name COMPILE;	N/A. Users should drop the procedure and recreate it.
DROP PROCEDURE sp_name	DROP PROCEDURE sp_name

Recommendations:

In Oracle, the procedural language is built in Oracle engine. On the other hand, DBMaster uses the ESQL/C for ESQL/C stored procedure or Java for Java stored procedure to do the coding. But from 5.2 version DBMaster can support SQL SP (Script Stored Procedure). This is the biggest difference between these two provided stored procedures (ESQL/JAVA). PL/SQL is the core components of Oracle. PL/SQL includes the commands that can create the logic store cells. PL/SQL could be used to add and manage data or the other database objects. If PL/SQL commands were stored in the Oracle, it would be referred to as “Stored Procedure”.

To develop ESQL/C stored procedures, DBMaster has to hook up to the external C-Compiler. This compiler is usually VC in Windows Platform, GCC in Linux. The normal process to build a C-Compiler in DBMaster is: compile the stored procedure, put it into the corresponding folder, and create procedure in dmsqlc with the syntax “create procedure from …” syntax. As to the coding in Procedural Language in Oracle or DBMaster, that is out of scope in this document. Users could read the **ESQL C Programmer’s Guide** for details.

For Java stored procedures, if you know how to access to Database using Java programs, the coding and creating processes are very easy and fast.

5.4.12 ORACLE AND DBMASTER IN AP

Oracle	**DBMaster**
Supported driver	
JDBC/ODBC,Hibernate,Nhibernate,OLE DB	JDBC/ODBC,DCI, Ruby, Hibernate, Nhibernate,OLE DB
Connection String	
ODBC: "Driver={Microsoft ODBC Driver for Oracle};” “Server=Server Name;” “DBQ=DarabaseName;” “ Uid=Username;” “Pwd=Password;"	ODBC: "Driver={DBMaster 5.1 Driver};Database=Database name;uid=Username;Pwd=Password;"
OLE DB: “Provider = OraOLEDB.Oracle;” “Data Source = DatabaseName;” “User Id = Username;” “Password = Password;”	OLE DB: "Provider=DMOLE51;Data Source= Databasename;User Id= Username; Password =;"
JDBC: Class.forName (“oracle.jdbc.driver.OracleDriver”).newInstance(); url = “jdbc:oracle:thin: @IP_Adress:protnumber:DatabaseName”; Connection conn= DriverManager.getConnection(url,user,password) ;	JDBC: Class.forName("dbmaster.sql.JdbcOdbcDriver "); Connection conn= DriverManager.getConnection (jdbc: dbmaster:// IP_Address:TCP_Port /DatabaseName, user, password);

5.5 System Tables

Each database has its system tables. Users may need query these tables to get some information.

We list three of them as followings:

Oracle	DBMaster
Check one table exist	
select 1 from tabs where table_name= ‘XXXXX’	select 1 from systable where table_name=’ XXXXX’
Check DB Version from SQL	
Select version FROM Product_component_version Where SUBSTR(PRODUCT,1,6)='Oracle'	select value from sysinfo where info='VERSION'
Check Procedure exist	
select count(*) from user_source where type= 'PROCEDURE' and name=’XXXXXXX’	SELECT COUNT(*) FROM sysprocinf WHERE modulename = ‘XXXXXXX’

6. DB Object Migration procedures

6.1 Schema and Date Migration

Please refer to *chapter 4* for more information about how to migrate a database from Oracle to DBMaster.

You should rebuild indexes, constrains and so on after your migration.

6.2 Convert UDF

There are huge differences between Oracle and DBMaster UDF. Please read the detailed introduction about it in *chapter 5 sections 5.4.9*.

First, we should analyze the UDF function in Oracle.

Second, we can rewrite UDF according to the syntax of DBMaster.

Note: Please spend some time for a careful technical evaluation before using UDF.

6.3 Convert Trigger

There are three kinds of triggers in Oracle, including DML, DDL, or logon trigger. DBMaster only include DML Triggers. This article will focus on (DML) triggers.

Difference in the Trigger between Oracle and DBMaster had been introduced in *chapter 5 section 5.4.10* which include syntax levels, trigger events, ”for each row/statement” syntax, ”after/before” syntax and so on.

First, we should analyze the Oracle Trigger.

Second, we can rewrite Trigger according DBMaster syntax.

Note: some syntax which we can’t support should be replaced with other methods. For example: we think to write a stored procedure for the some processes.

6.4 Convert Stored Procedure

Detailed Recommendations for stored procedures between Oracle and DBMaster have been introduced in *chapter 5.4.11*. **Stored Procedures and Stored Functions**. Here we mainly discuss how to convert stored produces from Oracle to DBMaster successfully. Oracle stored procedures use the PL/SQL but DBMaster uses the ESQL/C for ESQL/C stored produces or java for java stored procedures to do coding. PL/SQL includes the commands that can create the logical store cells. DBMaster can create logical store cells with SQL SP in release 5.2. And in current DBMaster version, we can develop ESQL/C stored procedures with external C-Complier or Java stored produces.

Because the difference is so big as above description, we can’t convert them directly. So we should do following things step by step.

Because the difference is so big as above description, we can't convert them directly. So we should do following things step by step.

1. First, we have to analyze the purpose of the stored procedures created by PL/SQL in Oracle.
2. Next, we need choosing one language from ESQL/C and Java for creating stored procedure.
3. Rewriting the stored produces with suitable syntax for DBMaster and make it have same action as the old in Oracle.
4. Creating and testing the stored procedure in DBMaster.

Note: For more details about creating stored procedures by ESQL/C or JAVA, please refer to the *ESQL C Programmer's Guide* or *Creating Stored procedures using Java* section in *DBA manual*.

7. AP migration procedures

It's very important for us to check application program interfaces first. For example, we should check whether the interface is supported by DBMaster if we want to migrate them from another database.

Next, we must consider how to rewrite the connect strings according to the driver.

Finally, mark the special syntax in Oracle and find the solution for DBMaster.

7.1 AP interface and Connect string

We must make clear what kinds of interfaces are used in application programs and whether these interfaces are supported by DBMaster.

What types of data provider or divers are used to access data source. JDBC, ODBC or any others, for example: If data provider changes, we might consider changing driver.

We can discuss each tier from following aspects.

Finally, you'd better do a quick testing for the application program that has been modified. In order to make sure it can connect to DBMaster successfully.

7.1.1 AP IN CLIENT

A part of application program codes that related to database connection or manipulation may need to do some modifying. Such as DSN, CONNECT SRTING and so on in client.

In addition, if the application need get some information from SYSTEM Table (or CATALOG). Please refer to the *chapter 5.5* to modify the usage.

7.1.2 MIDDLE-TIER

If use COM+ or implement DB-tier encapsulation implemented with similar technology, users need to consider modifying connect string and any other parameters of COM components in DB-tier.

7.1.3 AP OR (WEB) SERVER

Regarding AP server, users may need to modify some parameters that related to DB Server such as Server IP address, Port Number, Driver etc.

7.1.4 AP IN SERVER

Here, users need to check whether there are some schedules or tasks deployments exist in server separately and whether these programs need modifying.

7.2 Oracle special syntax and feature

On one hand, we must solve connect situation, on the other hand, we must pay a special attention to special syntax in Oracle. Consider what method is substitute for these special grammars.

There are too many special syntax exists in Oracle. In order to find replaced solutions for an alternative. We give some simple samples as followings. You must understand this aspect of knowledge about Oracle and DBMaster before migration. You also can Comparison with *chapter 5* that describes the difference between Oracle and DBMaster. For more information you can reference *Oracle and DBMaster User Guide*.

7.2.1 FOR SELECT STATEMENT

In Oracle, select statement must include “from” keyword. For example, you should write “select sysdate from dual”, ”from” keywords can not omitted, but in DBMaster, you only write “select curdate ()”. So if select statement includes “from dual” keywords we must remove it and make statement compatible when you transfer it to DBMaster.

7.2.2 FOR “ROWNUM” USAGE

In Oracle, you can write “select columns_name from table_name where ROWNUM<= count”, but in DBMaster, “ROWNUM” isn’t supported, we must use “select columns_name from table_name limit count” to realize the same function.

7.2.3 FOR NESTED QUERY

Suppose we have a table named tb_nest which record all staff information .If we want to know who the latest one for each department.

In Oracle, we can write following statements

```
select * from tb_nest t1
where come_date >= (select max(come_date) from tb_nest tb2 where tb2.dept = t1.dept)
```

In DBMaster, the grammar isn’t supported. In order to achieve the same function in DBMaster, we adopt the method of temporary table by rewriting statements.

```
select emp_from, max (come_date) as come_date from tb_nest group by emp_from
into temp;
select * from tb_nest tb1 join temp tb2 on tb1.emp_from =
tb2.emp_from and tb1.come_date=tb2.come_date
```

8. Testing application with new DB

Testing applications are required at any moment, at the beginning, in the process or at the end of migration. It can help us confirm our modifications or adjustments to be befitting.

8.1 How to pre-run for skip any object

In order to find problems timely and get to know where the problems exist, we must test the program every time to find out which part has something wrong.

It's better for us to begin migrating next section after having tested and ensured the part of you just finished has no problems. This is very helpful for you to migrate all applications programs from Oracle to DBMaster is successfully.

8.2 Test application with DBMaster after migration

A validate testing is required after whole application programs have been migrated completely from Oracle to DBMaster. You can ensure the application run normally on new platform with the validate testing.

9. Performance tuning

When you develop an application system with any database, the system performance is an important thing and you must be concerned about it. We must tune database after migration and make sure the application program run efficiently. Of cause, performance tuning is about the whole processes of using database not only tuning after migration .The amount data is growing in database. You should pay attention to database performance tuning often. If any database performance down, we should detect database and adjust timely in use.

Performance tuning need adjusting not until migration finished from Oracle to DBMaster. It's from the beginning design and planning the whole db to the end use.

Generally speaking, there are many factors affecting the performance of DBMaster. We can see them from the following figure.

Application System	Query Optimization
	Concurrent Process
	Application System Architecture
	Database Model Design (Tablespace, Table, Index, stored command, Stored procedure, Trigger)
Database System	Daemon (Auto-commit, Checkpoint, Update statistic, Backup server, Replication)
	Memory Allocation
	Disk I/O (Database Data partition)
OS	(File system, Raid)
Hardware	Network
	I/O
	Memory
	CPU

9.1 Application

It comprises writing queries that limit the use of stored commands or searches for procedures. Designing a good schema or developing an application with better utilities can both significantly increase applications performance.

Using indexes can improve the application performance for accessing to database if you built the index reasonable. For example, if you build some indexes only on the required columns in a table. DBMaster will find the data effectively.

Another attention for Applications is Concurrent Processes. Obviously, minimizing lock contention and avoiding deadlocks can increase throughput of applications. In addition, shortening transactions can promote concurrency, but it is possible to degrade database performance oppositely.

9.2 Database System

It includes **Disk I/O**, **Memory Allocation** and **Daemon**. Make sure there are enough physical memory for DCCA and few I/O access times.

9.2.1 TUNING MEMORY ALLOCATION

DBMaster stores information temporarily in memory buffers and permanently on disk. Since it takes much less time to retrieve data from memory than disk, performance will increase if data can be obtained from the memory buffers. The size of database memory allocation will affect performance of a database. However, performance will become an issue only if there is not enough memory. So we must tune the memory usage for a database and it includes how to calculate the required DCCA size, and how to monitor and allocate enough memory for the page buffers, journal buffers and system control area.

To achieve the best performance, follow the steps in the order shown:

1. Tune the operating system.
2. Tune the DCCA memory size.
3. Tune the page buffers.
4. Tune the journal buffers.
5. Tune the SCA.

Memory requirement for DBMaster varies according to the applications in use, tune memory allocation after tuning application programs and SQL statements.

9.2.1.1 Tuning an Operating System

The operating system should be tuned to reduce memory swapping and ensure that the system runs smooth and efficiently.

Memory swapping between physical memory and the virtual memory file on disks takes a significant amount of time. It is important to have enough physical memory for running processes. Measure the status of an operating system with the operating system utilities. An extremely high page-swapping rate indicates that the amount of physical memory in a system is not large enough.

In this case, you shoule remove any unnecessary processes or add more physical memory to the system.

9.2.1.2 Tuning DCCA Memory

The Database Communication and Control Area (DCCA) is a group of shared memory allocated by DBMaster servers. Every time DBMaster is started, it allocates and initializes the DCCA.

The DCCA is the resource most frequently accessed by DBMaster processes. It is important to ensure there is enough physical memory to prevent the operating system from swapping the DCCA to disks too often or it will seriously degrade performance of a database.

Usually a larger number of buffers are better for system performance. However, if the DCCA is too large to fit in physical memory, the system performance will degrade. Therefore, it is important to allocate enough memory for the DCCA but still fit the DCCA in physical memory.

You can set the appropriate parameters **DB_NBufs**, **DB_NJnlB** and **DB_ScaSz** in **dmconfig.ini** before starting the database to configure the size of each of the DCCA components.

The total memory allocation for the DCCA is the sum of the size of **DB_NBufs**, **DB_NJnlB** and **DB_ScaSz**.

9.2.1.3 Tuning Page Buffer Cache

DBMaster uses the shared memory pool for the data page buffer cache. The buffer cache allows DBMaster to speed up data access and concurrency control. Adjusting the size of the page buffers will have the greatest effect on performance.

We can improve buffer cache performance by following ways

1. Update statistics on schema objects.
2. Set NOCACHE on large tables.
3. Reorganize data in poorly clustered indexes.
4. Enlarge cache buffers.
5. Reduce the effect of checkpoints.

For concrete realization of above methods please reference *DBA manual Chapter Performance Tuning*.

9.2.1.4 Tuning Journal Buffers

The journal buffers store the most recently used journal blocks. With enough journal buffers, the time required to write journal blocks to disks and roll back transactions when updating data and reading journal blocks from disks is reduced.

You should determine whether there are sufficient journal buffers for the system. The optimum number of journal buffers is the sum of journal blocks needed by the longest running transactions at the same time.

There are two ways used to estimate the number of journal buffers, one is the number of used journal blocks and the other measurement is the journal buffer flush rate.

More details please reference DBA *manual Chapter Performance Tuning*.

9.2.1.5 Tuning the SCA

Cache buffers and some control blocks, such as session and transaction information, have a fixed size, and are pre-allocated from the DCCA when a database is started. However, some concurrency control blocks are allocated dynamically from the DCCA while the database is running, their size is specified by DB_ScaSz.

If a database application gets the error message “database request shared memory exceeds database startup setting”, it means that DBMaster cannot dynamically allocate memory from the SCA area. Usually, this error is due to a long transaction using too many locks. If this situation happens often, solve it with the methods illustrated below.

1. Avoid Long Transactions

2. Avoid Excessive Locks on Large Tables

3. Increase the SCA size

For details please reference DBA *manual Chapter Performance Tuning*

9.2.2 QUERY OPTIMIZATION

The query optimizer will make a query of SQL commands much faster and efficient by means of choosing the best execution method internally.

If performance degrades, we should check the query plan by the command “Set dump plan on” and the SQL to improve the performance by forcing index scan, rewriting query, etc. For details please reference DBA *manual Chapter Performance Tuning*.

9.3 OS

A suitable OS is important for improving the performance of whole system, so please chose one OS with special designed for supporting the application disposal and the database as possible as you can.

In addition, about hard disks which support the technical Raid, please chose different Raid Level for different data types. For example, in DBMaster, you can put data file into Raid 1,3,5, and put journal file into Raid 0,which can guarantee safeness and a high efficiency.

9.4 Hardware

It is the basic factor not only affects the performance of DBMaster, but also affects the whole PC’s.

- **CPU:** A faster CPU or multi CPUs can help improving performance.
- **Memory:** Enough memory can hold more cached data, so I/O access time will be reduced.
- **I/O:** Faster hard disks can improve the I/O throughput and more hard disks can promote the I/O concurrency.
- **Network:** Speeding up transmission for network can reduce response time for users. Using only network protocols required will reduce load balancing of the operating systems.

  Obviously, enhancing the hardware can greatly improve the overall database system performance absolutely.

On the whole, we must rebuild indexes, adjust configuration according to DB, AP, and so on which in order to improve the database application program performance. For more contents please refer to the *DBA manual chapter Performance Tuning*.

10. Appendix – Migration Samples

In this chapter, we will provide some real samples for both DBMaster and Oracle. The content involves samples for not only some Table Schema and Data but also applications with different program languages. It provides a good demonstration of Migration from Oracle to DBMaster.

The purpose is to help users quickly get to know the difference between DBMaster and Oracle, and easily catch on the migration steps. It can reduce the migration costs.

In addition, these simple samples can not contain all of instances at present. And we will enhance all the features which the users care in this document continually.

10.1 Table Schema for all Types

In order to make users get to know Types Mapping between Oracle and DBMaker, we give an example here. Users can write the SQL manually or generate the Script files automatically by **JDatatransfer Tool**.

In this section, we don't refer the migration of DATA, and we will demonstrate the samples for migration of ordinary types and special types data in next *chapter 10.2*.

10.1.1 CREATE TABLE WITH ALL TYPES IN ORACLE

Some types are not supported in Oracle, and they will be converted into some corresponding types automatically while creating the table. For example: SMALLINT, DECIMAL, INTEGER and REAl.

```
create table oracle_all_types(
 col_smallint       smallint,
 col_decimal        decimal(13,3),
 col_varchar        varchar(30),
 col_varchar2       varchar2(30),
 col_raw            raw(200),
 col_char           char(30),
 col_nchar          nchar(40),
 col_date           date,
 col_long           long,
 col_blob           blob,
 col_clob           clob,
 col_nclob          nclob,
 col_bfile          bfile,
 col_rowid          rowid,
 col_number         number(8,2),
 col_integer        integer,
 col_float          float,
 col_real           real);
```

After the table being created, you can find the following conversions.

SMALLINT ➔ NUMBER(38,0)

DECIMAL(13,3) ➔ NUMBER(13,3)

INTEGER ➔ NUMBER(38,0)

REAl ➔ FLOAT(63)

10.1.2 MODIFY TABLE SCHEMA MANUALLY

DBMaster doesn't support the FLOAT type, and convert it into DOUBLE automatically.

```
create table oracle_all_types (
 col_smallint              smallint,
 col_decimal               decimal(13, 3),
 col_varchar               varchar(30),
 col_varchar2              varchar(30),
 col_raw                   binary(200),
 col_char                  char(30),
 col_nchar                 nchar(40),
 col_date                  timestamp,
 col_long                  long varchar,
 col_blob                  long varbinary,
 col_clob                  clob,
 col_nclob                 nclob,
 col_bfile                 long varbinary,
 col_rowid                 char(18),
 col_number                decimal(8, 2),
 col_integer               integer,
 col_float                 float,
 col_real                  real);
```

10.1.3 MIGRATE WITH JDATATRANSFER TOOL

In addition, if you migrate Table Schema from Oracle with **JDatatransfer Tool**, some columns will be converted to different types by above steps manually.

```
create table  ORACLE_ALL_TYPES (
 COL_SMALLINT     DECIMAL(38, 0)     default null ,
 COL_DECIMAL       DECIMAL(13, 3)    default null ,
 COL_VARCHAR      VARCHAR(30)        default null ,
 COL_VARCHAR2     VARCHAR(30)        default null ,
 COL_RAW          BINARY(200)        default null ,
 COL_CHAR         CHAR(30)           default null ,
 COL_NCHAR        NCHAR(40)          default null ,
 COL_DATE         TIMESTAMP          default null ,
 COL_LONG         LONG VARCHAR       default null ,
 COL_BLOB         LONG VARBINARY     default null ,
 COL_CLOB         LONG VARCHAR       default null ,
 COL_NCLOB        NCLOB              default null ,
 COL_BFILE        LONG VARBINARY     default null ,
 COL_ROWID        VARCHAR(10)        default null ,
 COL_NUMBER       DECIMAL(8, 2)      default null ,
 COL_INTEGER      DECIMAL(38, 0)     default null ,
```

```
COL_FLOAT       DOUBLE          default null ,
COL_REAL        DOUBLE          default null );
```

Note: *Please modify VARCHAR(10) to CHAR(18) for **ROWID Type**.*

10.2 Table Schema and Data

In this section, we will divide all the Data Types into ***Ordinary Type*** and ***Special Type***.

The ***Ordinary Type*** data is ordinary characters and the numeric data type, which can be exported with *TEXT-Format* file from Oracle via ***SQL Developer Tool***, and imported into DBMaster via ***Import from Text*** in ***JDataTransfer Tool*** (or via manual ***import*** command).

The ***Special Type*** data have different structures for different Databases, which must be converted by some built-in functions or ODBC Applications. In addition, some of Data Types (CLOB, NCLOB, BLOB) cannot be exported from Oracle via ***SQL Developer*** tool.

10.2.1 ORDINARY CHARACTER AND NUMERIC DATA TYPE

Step 1: Create table *ordinary_types* in Oracle.

```
create table ordinary_types(
 col_smallint        smallint,
 col_decimal         decimal(13,3),
 col_varchar         varchar(30),
 col_varchar2        varchar2(30),
 col_char            char(30),
 col_long            long,
 col_number          number(8,2),
 col_integer         integer,
 col_float           float,
 col_real            real);
```

Step 2: Insert Data by some Applications or by hand.

For example:

```
insert into ordinary_types values(100,3456.4,'col_varchar','col_varchar2','char30','long varchar',
345435.22,10000,454.23,34535.56);

insert into ordinary_types values(200,3456.4,'col_varchar','col_varchar2','char30','long varchar',
345435.22,20000,454.23,34535.56);

insert into ordinary_types values(300,3456.4,'col_varchar','col_varchar2','char30','long varchar',
345435.22,30000,454.23,34535.56);
```

Step 3 (recommend): Import from ODBC in DBMaster.

Please refer to *Chapter 4.1 Database transfer tools*. Import the table and data by the default choice.

Check the table and data in dmSQL, JSQL or JDBA tool:

For example:

```
dmSQL> def table ordinary_types;
dmSQL> select * from ordinary_types;
```

Optional step (step 3): Create table *ordinary_types* in DBMaster.

If you don't want to import both tables and data from ODBC, you can create table *ordinary_types* in DBMaster at first.

```
create table ordinary_types(
 col_smallint        smallint,
```

```
col_decimal          decimal(13,3),
col_varchar          varchar(30),
col_varchar2         varchar(30),
col_char             char(30),
col_long             long varchar,
col_number           decimal(8,2),
col_integer          integer,
col_float            float,
col_real             real);
```

Optional step (step 3--1): Export Data separately from Oracle.

Please refer to *Section 4.1.2 Oracle SQL Developer Tool* and export the Data with TEXT formats from Oracle.

Optional step (step 3--2): Import Data into DBMaster.

Please import into DBMaster via **JDATA Transfer Tool** which described in *chapter 4.1.2.2* and you must choose the uniform separator character to export TEXT format data. For example: Comma (Semicolon or Vertical Bar) for Column Delimiter, {CR}{LF} for Row Delimiter.

In addition, we recommend users to use the IMPORT command in dmSQL tools as following:

```
dmSQL> import ordinary_types from c:\test\ordinary_types.txt description c:\test\desc.txt;
```

desc.txt

```
FORMAT=VARIABLE
COLUMN_DELIMITER='\t'
ROW_TERMINATOR="\r\n"
QUOTATION=DOUBLE_QUOTE
ESCAPE_CHAR=YES
START_WITH_ROW=1
```

10.2.2 SPECIAL DATA TYPE

Step 1: Create table *special_types* in Oracle.

```
create table special_types(
 col_raw              raw(200),
 col_nchar            nchar(40),
 col_date             date,
 col_blob             blob,
 col_clob             clob,
 col_nclob            nclob,
 col_bfile            bfile,
 col_rowid            rowid);
```

Step 2: Insert Data by some Applications or by hand.

For example:

```
insert into special_types values('4100450045004100','42004200420042004300430043004300',to_date('2009-5-21
18:55:49','yyyy/mm/dd HH24:MI:SS'),'41004500','clob - long
varchar','414243444546474849 4a',bfilename('BDUMP_DIR','sqlplus.exe'),chartorowid('AAAADDAAEAAAAGrAAA'));

insert into special_types values('4100450045004100','42004200420042004300430043004300',to_date('2009-5-21
18:55:49','yyyy/mm/dd HH24:MI:SS'),'41004500','clob - long
varchar','414243444546474849 4a',bfilename('BDUMP_DIR','sqlplus.exe'),chartorowid('AAAADDAAEAAAAGrAAA'));

insert into special_types values('4100450045004100','42004200420042004300430043004300',to_date('2009-5-21
18:55:49','yyyy/mm/dd HH24:MI:SS'),'41004500','clob - long
varchar','414243444546474849 4a',bfilename('BDUMP_DIR','sqlplus.exe'),chartorowid('AAAADDAAEAAAAGrAAA'));
```

Note: *Before version 11g, Oracle don't support display the some Types (CLOB/BLOB/BFILE) in SQL-PLUS.*

Step 3: Create Table in DBMaster manually or ***Export from ODBC***.

You can only export table schema from ODBC via JDataTransfer Tool, Please refer to the ***Sub step 4*** in *4.1.1.2 Execute steps Import from ODBC* and choose ***Create destination table***.

Note: *Please modify VARCHAR(10) to CHAR(18) for **ROWID Type***.

Certainly, you can create table manually with the following table schema.

```
create table special_types (
 col_raw          binary(200),
 col_nchar        nchar(40),
 col_date         timestamp,
 col_blob         long varbinary,
 col_clob         clob,
 col_nclob        nclob,
 col_bfile        long varbinary,
 col_rowid        char(18));
```

Step 4: Import the special type Data into DBMaster.

Please use **JDataTransfer Tool** in DBMaster and refer to the Chapter *4.1.1.2 Execute steps Import from ODBC*, and modify ***Transform*** after choosing ***Source Table.***

(Please refer to the following Steps and Chart)

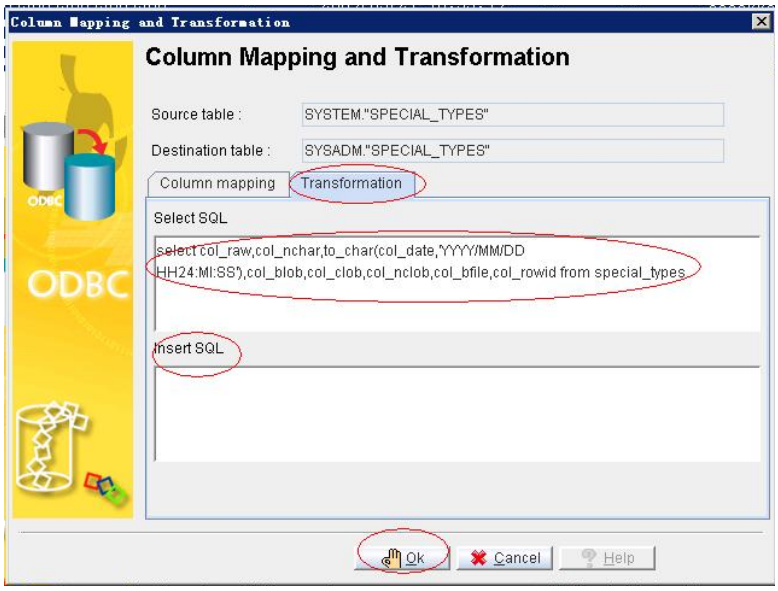
Step A: Click on the Tab Transformation and input the ***Select SQL*** for getting the result from Oracle.

```
select col_raw, col_nchar, to_char(col_date, 'YYYY/MM/DD HH24:MI:SS'), col_blob, col_clob, col_nclob, col_bfile, col_rowid
from special_types
```

Step B: Input the ***Insert SQL*** for Inserting into DBMaster.

```
insert into special_types(col_raw, col_nchar, col_date, col_blob, col_clob, col_nclob, col_bfile, col_rowid)
values(?, ?, ?, ?, ?, ?, ?, ?)
```

If ***Insert SQL*** includes all the columns, needn't inputting (as following Chart).



Note: Only ***DATE Type*** need being formatted by the built-in function *TO_CHAR()*, all other special types can be imported into DBMaster by default steps through ***Import from ODBC*** which are same as ordinary Data Type.

10.3 Applications (Source Code segment)

We provide some parts of Source Code segments in this section. And the issue is focusing mainly on the different usage of ***Connection*** between DBMaster and Oracle.

In addition, we will demonstrate the different usage of ***placeholder*** in JAVA and C# Language Samples. The placeholder in JAVA is "**?**" when users pass parameters; The placeholder in C# is same "**?**" for DBMaster, and is "**:**xxxxx" for Oracle (*ODP.NET--Oracle Data Provider for .NE*T).

10.3.1 JAVA LANGUAGE

- Oracle

```
try{
    ......
    Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
    Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@192.168.0.8:1521:testdb",
"system", "oracle");
    PreparedStatement pstmt = conn.prepareStatement("insert into ordinary_types(col_varchar,
col_integer, col_float) values(?,?,?)");
    pstmt.setString(1, "varchar-abcbc");
    pstmt.setInt(2, 1000);
    pstmt.setFloat(3, (float)32322555.3332);
    pstmt.executeUpdate();
    ......
    }
}catch(Exception ex){
    ex.printStackTrace();
}
```

- DBMaster

```
try{
    ......
    Class.forName("dbmaster.sql.JdbcOdbcDriver").newInstance();
    Connection conn = DriverManager.getConnection("jdbc:dbmaster:testdb","SYSADM","test123");
    PreparedStatement pstmt = conn.prepareStatement("insert into ordinary_types(col_varchar,
col_integer, col_float) values(?,?,?)");
    pstmt.setString(1, "varchar-abcbc");
    pstmt.setInt(2, 1000);
    pstmt.setFloat(3, (float)32322555.3332);
    pstmt.executeUpdate();
    ......
    }
}catch(Exception ex){
    ex.printStackTrace();
}
```

Note: DBMaster only supports JDBC Type2 at present, users need to install native DLL for JDBC Driver. In addition, don't forget to set IP and Port in Dmconfig.ini.

10.3.2 C# LANGUAGE

- Oracle (*ODP.NET--Oracle Data Provider for .NE*T)

```
String connStr = "Data Source=testdb;User Id=system;Password=oracle;";
OracleConnection conn = new OracleConnection(connStr);
```

```
conn.Open();
OracleCommand cmd = new OracleCommand();
cmd.Connection = conn;
cmd.CommandText = "insert into ordinary_types(col_integer, col_char, col_float) values(:p1,:p2,:p3)";
cmd.Parameters.Add(new OracleParameter(":p1",OracleType.Int32));
cmd.Parameters.Add(new OracleParameter(":p2", OracleType.Char, 30));
cmd.Parameters.Add(new OracleParameter(":p3", OracleType.Number));
cmd.Parameters[":p1"].Value = 1001;
cmd.Parameters[":p2"].Value = "Li Ping";
cmd.Parameters[":p3"].Value = 2345.34;
cmd.ExecuteNonQuery();
cmd.Parameters.Clear();
cmd.CommandText = "select col_integer, col_char, col_float from ordinary_types where col_integer =1001";
cmd.CommandType = CommandType.Text;
OracleDataReader reader = cmd.ExecuteReader();
while(reader.Read()){
   for(int i=0;i<reader.FieldCount;i++){
      Console.WriteLine(reader[i]);
   }
}
conn.Close();
```

- DBMaster (*ADO.NET ODBC Provider*)

```
String connStr = "Driver={DBMaster 5.1 Driver}; Database=testdb; Uid=SYSADM; Pwd=test123";
OdbcConnection conn = new OdbcConnection(connStr);
conn.Open();
OdbcCommand cmd = new OdbcCommand();
cmd.Connection = conn;
cmd.CommandText = "insert into ordinary_types(col_integer, col_char, col_float) values(?,?,?)";
cmd.Parameters.Add(new OdbcParameter("p1",OdbcType.Int));
cmd.Parameters.Add(new OdbcParameter("p2",OdbcType.Char, 30));
cmd.Parameters.Add(new OdbcParameter("p3",OdbcType.Numeric));
cmd.Parameters[0].Value = 1001;
cmd.Parameters[1].Value = "Li Ping";
cmd.Parameters[2].Value = 2345.34;
cmd.ExecuteNonQuery();
cmd.Parameters.Clear();
cmd.CommandText = "select col_integer, col_char, col_float from ordinary_types where col_integer =1001";
OdbcDataReader reader = cmd.ExecuteReader();
while(reader.Read()){
   for(int i=0;i<reader.FieldCount;i++){
      Console.WriteLine(reader[i]);
   }
}
conn.Close();
```

Note: We don't provide the ***.NET Provider*** at present, so we only can use ***ADO.NET ODBC Provider*** or ***ADO.NET OLEDB Provider*** to connect DBMaster (The following *Source Code segment* is for *ADO.NET OLEDB Provider*).

```
String connStr = "Provider=DMOLE51; Data Source=testdb; User Id=SYSADM; Password=test123";
OleDbConnection conn = new OleDbConnection(connStr);
conn.Open();
```

```
OleDbCommand cmd = new OleDbCommand();
cmd.Connection = conn;
cmd.CommandText = “insert into ordinary_types(col_integer, col_char, col_float) values(?,?,?)”;
cmd.Parameters.Add(new OleDbParameter(“p1”,OleDbType.Integer));
cmd.Parameters.Add(new OleDbParameter(“p2”,OleDbType.Char, 30));
cmd.Parameters.Add(new OleDbParameter(“p3”,OleDbType.Numeric));
cmd.Parameters[0].Value = 1001;
cmd.Parameters[1].Value = “Li Ping”;
cmd.Parameters[2].Value = 2345.34;
cmd.ExecuteNonQuery();
cmd.Parameters.Clear();
cmd.CommandText = “select col_integer, col_char, col_float from ordinary_types where col_integer =1001”;
OleDbDataReader reader = cmd.ExecuteReader();
while(reader.Read()){
   for(int i=0;i<reader.FieldCount;i++){
      Console.WriteLine(reader[i]);
   }
}
conn.Close();
```

10.3.3 PHP LANGUAGE

We demonstrate the PHP PDO samples. If users don’t adopt PDO, please refer to our PHP samples in Installed Directory which use the PHP ODBC API.

- Oracle

```
<?php
 try{
    $dbh = new PDO("OCI:dbname=testdb;charset=UTF-8", "system", "oracle");
    /*** echo a message saying we have connected ***/
    echo 'Connected to database';
 }catch(PDOException $e){
    echo $e->getMessage();
 }
?>
```

- DBMaster

```
<?php
 try{
    $dbh = new PDO("odbc:Driver={DBMaster 5.1 Driver};Database=testdb", "sysadm", "test123");
    /*** echo a message saying we have connected ***/
    echo 'Connected to database';
 }catch(PDOException $e){
    echo $e->getMessage();
 }
?>
```

Note: If users didn’t use the PDO in Oracle as following:

```
<?php
$conn = ora_logon( “system@testdb” , “oracle” );
$mycursor ora_open($conn);
ora_parse ($mycursor, ’SELECT * FROM example’, 0);
ora_exec ($mycursor);
```