DBMaster

Reference Guide for PostgreSQL 8.4 Migration to DBMaster 5.1

Version: 01.00

Author: DBMaster Support Team and Research & Development Division, Syscom Computer Engineering CO.
Document No: 51/DBM51-T11152010-02-MRPS

Publication Date: November 15, 2010

Content

1. Overview 1

2. Analyze the current system 2

2.1 Analyze AP system 2
2.2 Analyze Database Objects 2

3. Setup migration environment 4

4. Methods for migrating table schema and data . 5

4.1 Database transfer tools 5
4.1.1 JDATATRANSFER TOOL IN DBMASTER 5
4.2 Other Third party tools 13
4.2.1 POSTGRESQL DATA WIZARD 14
4.2.2 SQL SCRIPT BUILDER 22
4.3 Modify DDL manually 25
4.4 Write code 26

5. Compare PostgreSQL and DBMaster 27

5.1 Schema Comparison 27
5.1.1 THE TERMINOLOGY COMPARISON 27
5.1.2 STORAGE STRUCTURE COMPARISON 28
5.1.3 PROCESS AND RELATED TERM DEFINITION 28
5.1.4 RESERVED WORD CONFLICT IN DATABASE OBJECT 29
5.1.5 DATABASE OBJECT DESIGN CONCERNS 30
5.2 Data Types Mapping 33
5.2.1 COMMON DATA TYPE MAPPING 33
5.2.2 DATA TYPES MAPPING CONCERN 37
5.3 Index Mapping 38
5.4 Data Manipulation Language (DML) 42
5.4.1 CONNECTING TO THE DATABASE 43
5.4.2 SELECT STATEMENTS 43
5.4.3 INSERT STATEMENTS 44
5.4.4 UPDATE STATEMENTS 44
5.4.5 DELETE STATEMENTS 45
5.4.6 OPERATORS 45
5.4.7 BUILT-IN FUNCTIONS 47
5.4.8 LOCKING CONCEPTS AND DATA CONCURRENCY ISSUES 50
5.4.9 TRIGGER DIFFERENCE 51

5.4.10 STORED PROCEDURES AND STORED FUNCTIONS 51
5.4.11 USER-DEFINED TYPES 52
5.4.12 PRIVILEGES 52
5.4.13 POSTGRSQL AND DBMASTER IN AP 53
5.5 System Tables 53

6. DB Object Migration procedures 55

6.1 SCHEMA AND DATE MIGRATION 55
6.2 CONVERT USER-DEFINED TYPES 55
6.3 CONVERT TRIGGER 55
6.4 CONVERT STORED PROCEDURE 55

7. AP migration procedures 57

7.1 AP interface and Connect string 57
7.1.1 AP IN CLIENT 57
7.1.2 MIDDLE-TIER 57
7.1.3 AP OR (WEB) SERVER 57
7.1.4 AP IN SERVER 57
7.2 PostgreSQL special syntax and feature 58
7.2.1 FOR SELECT STATEMENT 58
7.2.2 FOR INHERITANCE AND PARTITIONING 58
7.2.3 FOR NESTED QUERY 58

8. Testing application with new DB 59

8.1 How to pre-run for skip any object 59
8.2 Test application with DBMaster after migration .. 59

9. Performance tuning 60

9.1 Application 61
9.2 Database System 61
9.2.1 TUNING MEMORY ALLOCATION 61
9.2.2 QUERY OPTIMIZATION 63
9.3 OS 63
9.4 Hardware 63

10. Appendix – Migration Samples 64

10.1 Table Schema for all Types 64
10.1.1 CREATE TABLE WITH ALL TYPES IN POSTGRESQL 64
10.1.2 MIGRATE WITH JDATATRANSFER TOOL 65
10.1.3 MODIFY TABLE SCHEMA MANUALLY 66

10.2 Table Schema and Data 67
10.2.1 ORDINARY CHARACTER AND NUMERIC DATA TYPE 67
10.2.2 SPECIAL DATA TYPE 68
10.3 Applications (Source Code segment) 70
10.3.1 JAVA LANGUAGE 71
10.3.2 C# LANGUAGE 71
10.3.3 PHP LANGUAGE 73

1. Overview

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

PostgreSQL is known as a powerful, open source object-relational database product in industry at present. With this document, users easily understand the pros and cons between PostgreSQL and DBMaster. Users are also aware of the characteristics of both DBMaster and PostgreSQL.

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

We will introduce DBMaster in the following aspects:

- Migrate PostgreSQL database to DBMaster 5.1
- 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 types mapping rules.

2. Analyze the current system

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

For example, we should analyze current operating system and get to know what system we will use (Windows, Linux or Unix). 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 and users should know which one is 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 PostgreSQL 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 PostgreSQL
- The workaround of PostgreSQL special feature
- Special syntax of PostgreSQL
- 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 to evaluate how much space is required for storing data.

Next, we should analyze all tables' structures and get to know what contents will be stored in these tables. 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 PostgreSQL and DBMaster. So we should consider these differences in advance. We will introduce some aspects as followings:

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

- Data types belong to DBMaster but not apply to PostgreSQL.
- Built-functions belong to PostgreSQL but not apply to DBMaster.
- Built-functions belong to DBMaster but not apply to PostgreSQL.
- Indexes belong to PostgreSQL but not apply to DBMaster.
- Indexes belong to DBMaster but not apply to PostgreSQL.

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 PostgreSQL before doing anything. Then we will install DBMaster and create a database. Certainly, users should reserve enough disk space for DBMaster database db files at first. Here it's only for convenience, you can install DBMaster in the same machine with PostgreSQL.

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

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

- Adjust DBMaster configure parameters if necessary.
- Enroll settings in windows system.
- Special workaround for migration.
- DSN or environment variables should be set.
  - UnixODBC in Linux system (only Migration on Linux system)
  - ODBC Driver Manager in Windows
- Which DBMaster. (Normal version or bundle version).
- How to start the DB service? (via Dmserver or via 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 table schemas and data in this section and other aspects will be described in *chapter 6* and *chapter 7*.

4.1 Database transfer tools

PostgreSQL doesn't provide any tool of itself for migration, so we only introduce the JdataTransfer tool of DBMaster in this chapter.

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
- 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 introduce **Import from ODBC**. DBMaster supports importing data from other data sources via ODBC. Other data sources may include other database engines, such as PostgreSQL, 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 **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 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.

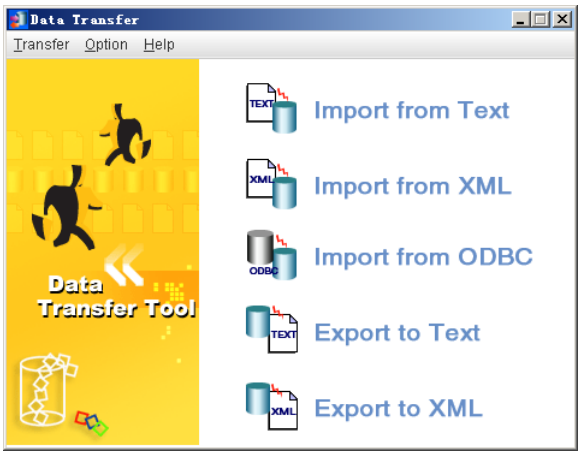
4.1.1.1 How to start the DBMaster JDataTransfer

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

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

4.1.1.2 Execute steps Import from ODBC

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



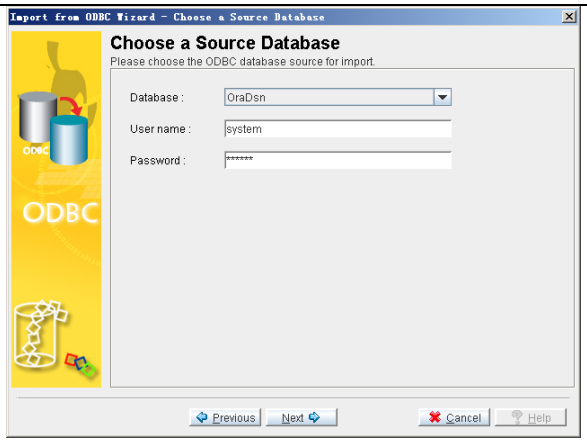
Step 2: selected **Import from ODBC** option and open the **Import from ODBC Wizard**.

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.

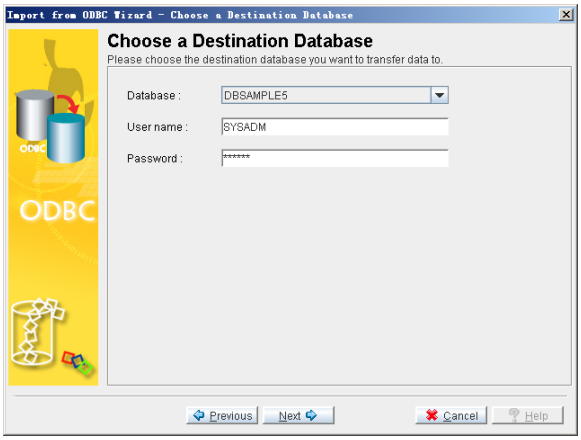


Step 4: Click on **Next**. The **Choose a Destination Data Source** window appears.

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.



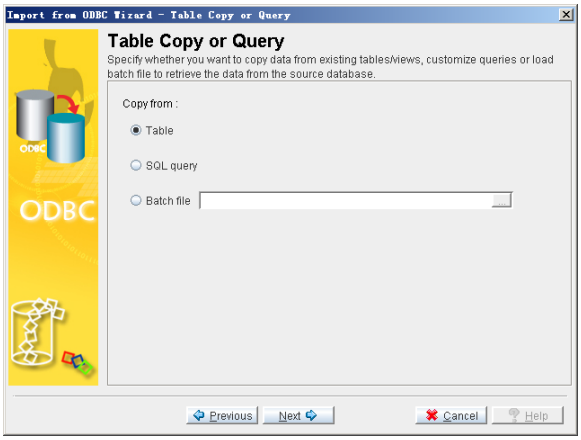
Step 5: Click on **Next**. The **Table Copy or Query** window appears.

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

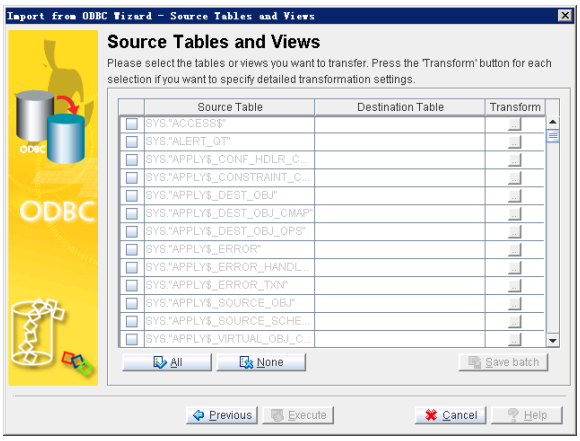
Batch file: To import data through an XML file.



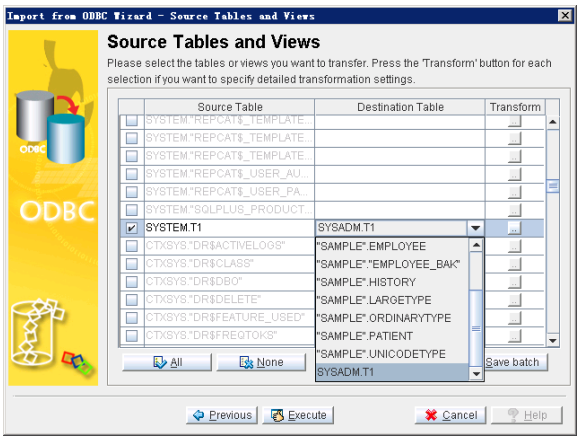
Three choices and corresponding operations:

➢ Selected" Table" check box

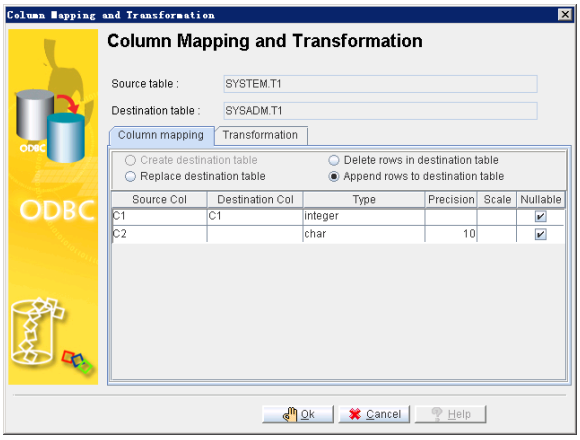
Sub_step 1: Click on **Next**. The **Source Tables and Views** window appears. All tables from the source database will appear in the Source Table column. Check the box to the left of each table to import.



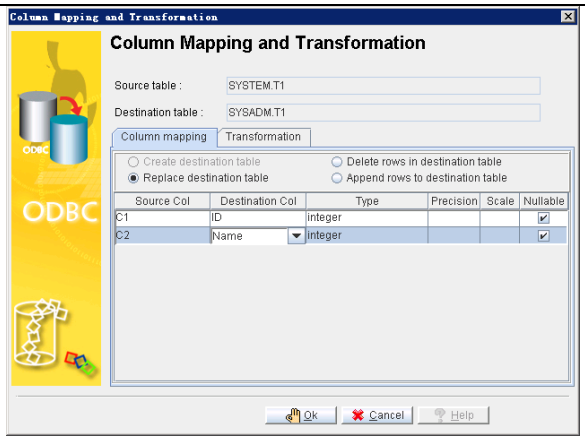
Sub_step 2: For each source table or view selected, click on the **Destination Table** field. If desired, change the name of the destination table by selecting a new table from the menu or entering a new name.



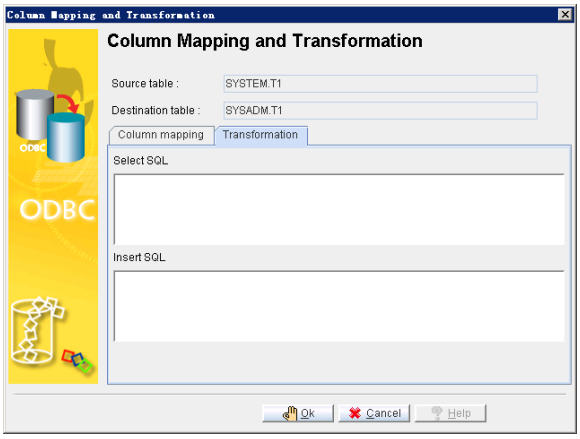
Sub_step 3: You can modify **Column mapping** or the result set to import by clicking on the **Transformation** button of the corresponding source and destination table.



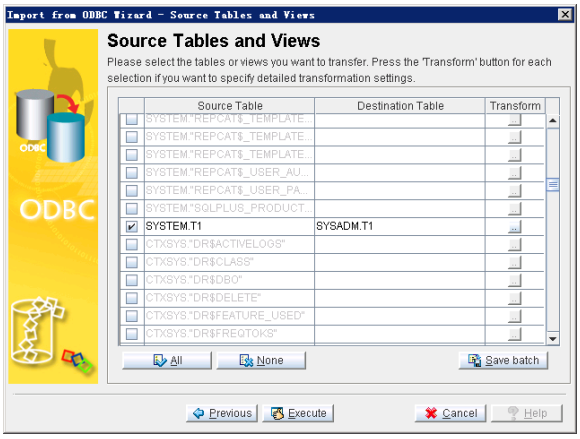
Sub_step 4: Change the name of the **Destination Column** by selecting a new column from the menu or entering a new name.



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.

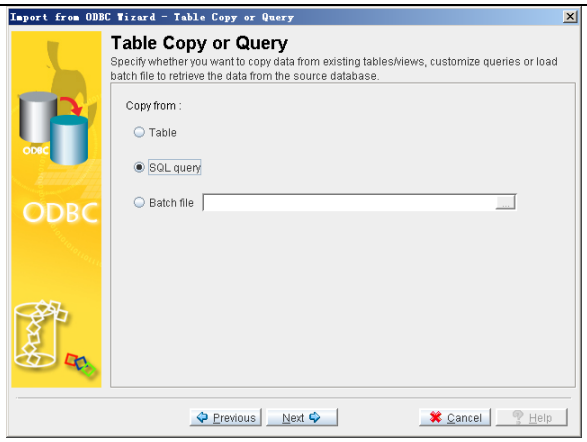


Sub_step 6: Click on **OK** to return to the **Source Tables and Views** window. You may also choose to save the map of the import ODBC schema to an XML file by clicking on **save batch**.

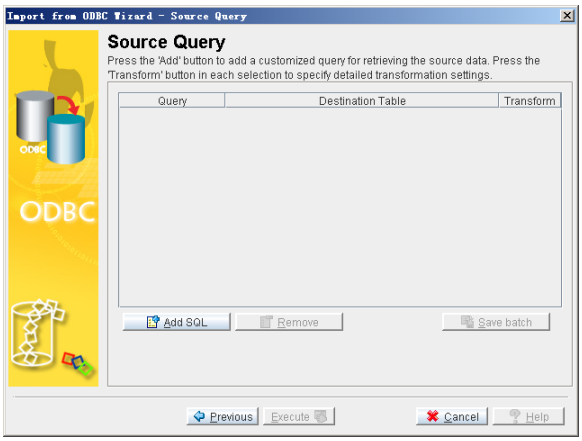


➢ Select “SQL query” check box

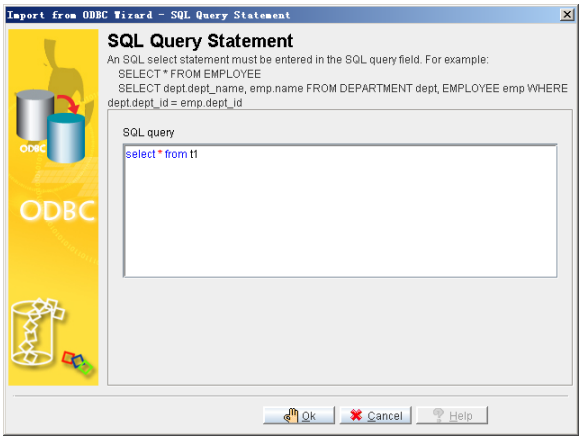
Sub_step 1: Click on **Next**. The **Table Copy or Query** window appears. Click on **Add SQL**. The **SQL Query Statement** window appears. And enter a valid SQL SELECT statement into the **SQL Query** field.



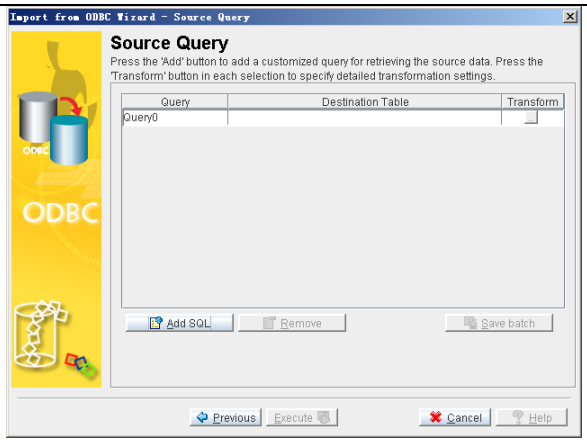
Sub_step 2: Click on **OK**. The **Source Query** window reappears.



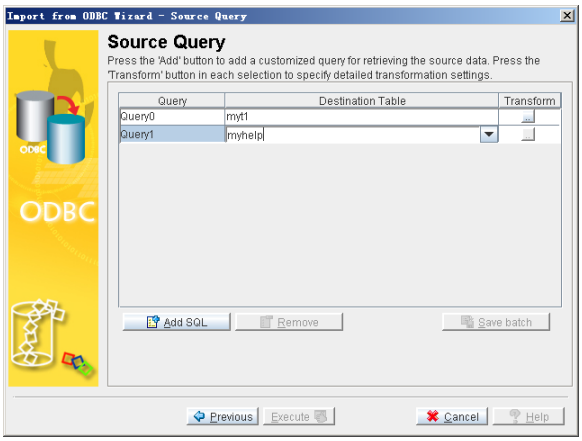
Sub_step 3: Click on **Add SQL**. The **SQL Query Statement** window appears. Enter a valid SQL SELECT statement into the **SQL Query** field.



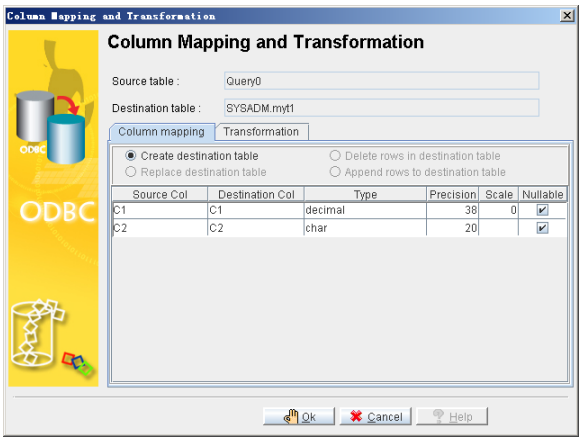
Sup_step 4: click on **OK** to return to **Source Query** page.



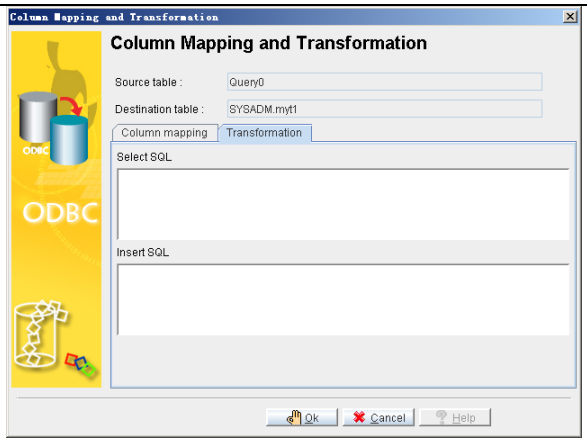
Sub_step 5: You can add more SQL query statements by clicking on **Add SQL** and change the name of the destination column by selecting a new column from the menu or entering a new name.



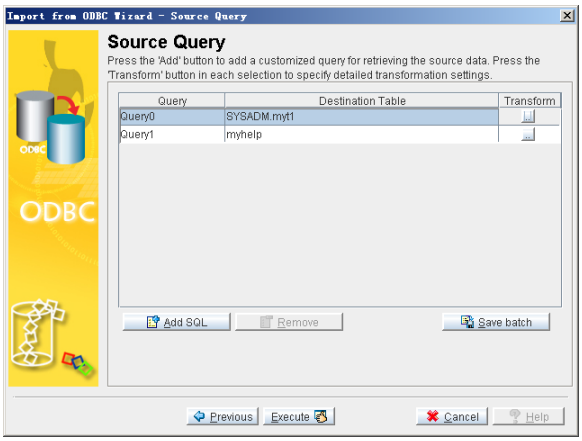
Sub_step 6: you also can modify the mapping of source and destination columns by clicking on the **Transformation** button.



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.

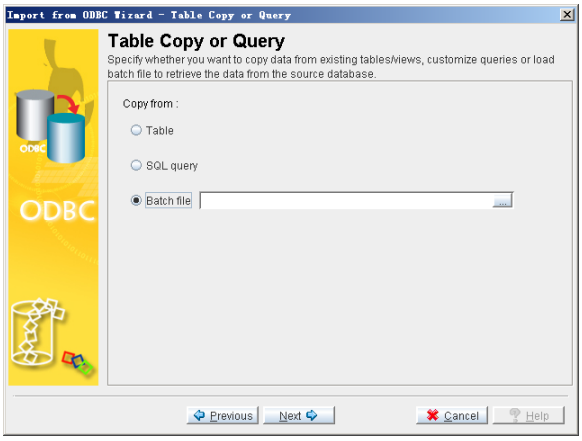


Sub_step 8: click on OK and return to **Source Query** page. You also can choose to save the map of the import ODBC schema to an XML file by clicking on **Save batch**. The Save Batch File will open. Select or create an XML file to save the imported ODBC map schema. Click on **Save Batch File** to create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

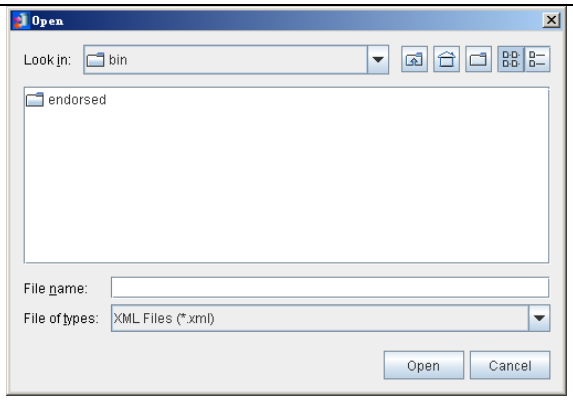


➢ Select “Batch file” Check box

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.

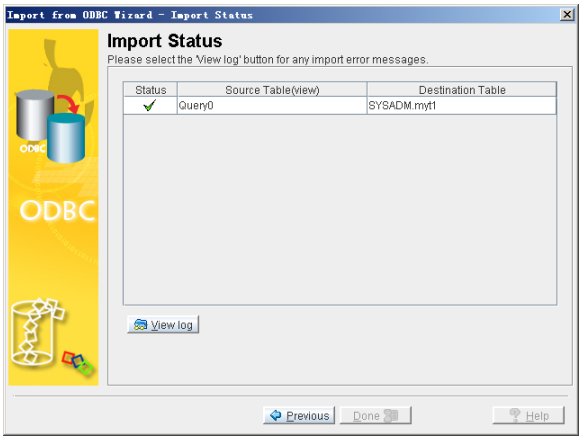


Sub_step 2: Click on **Next**. The **Source Query** window will open, displaying a mapping Schema according to the XML file.

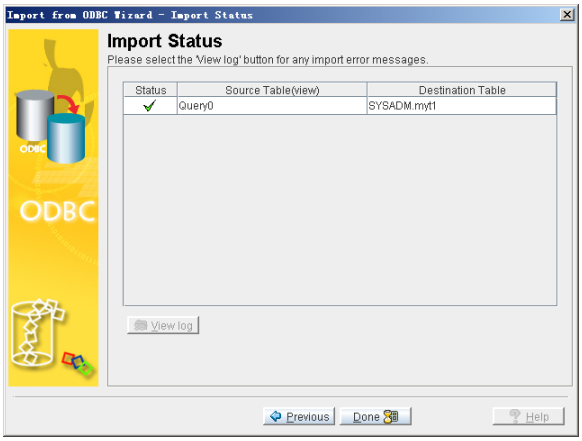


Note: The remaining operations are same as select **Table.**

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 error occurs, click on **done**.



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 POSTGRESQL DATA WIZARD

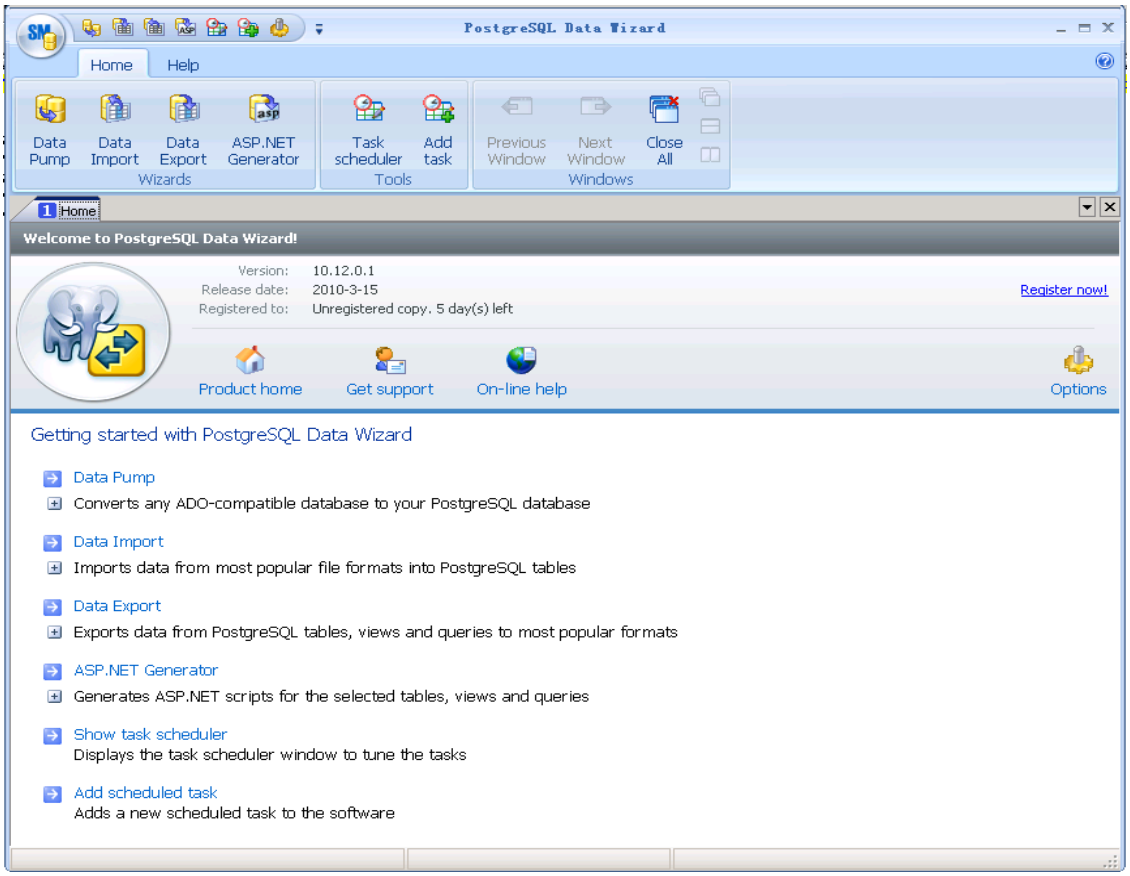
PostgreSQL Data Wizard is a powerful Windows GUI utility for managing your PostgreSQL data. It provides you with a number of easy-to-use tools for performing the required data manipulation easily and quickly. For example: Data Pump -- transfer any schemas and data to PostgreSQL; Data export to as many as 18 file formats; Data import from Excel, CSV, text files and more; ASP.NET Generator --create full set of ASP.NET scripts; Flexible Task Scheduler; The Agent application to execute tasks in background mode; Powerful command-line interface.

Here, we only especially introduce *Data export to CSV file formats*, which can prepare the TEXT-Format data for importing into DBMaster with ***JDataTransfer*** tool or ***Import*** command.

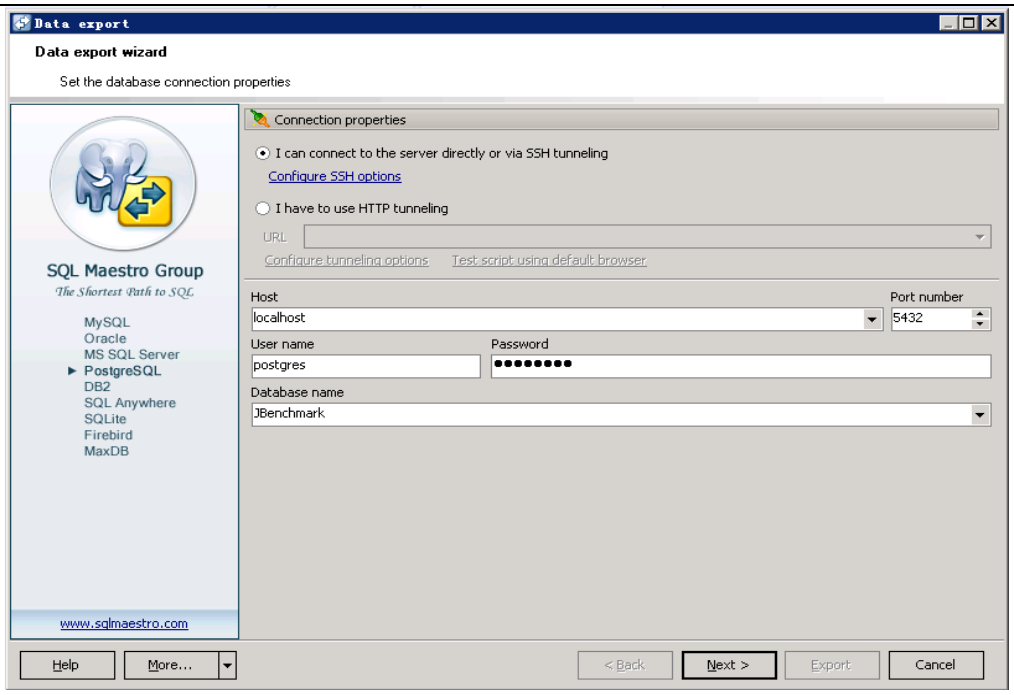
4.2.1.1 Export Data from PostgreSQL

Step 1: Start the tool in start menu

Start>programs>SQL Maestro Group>PostgreSQL Data Wizard>PostgreSQL Data Wizard

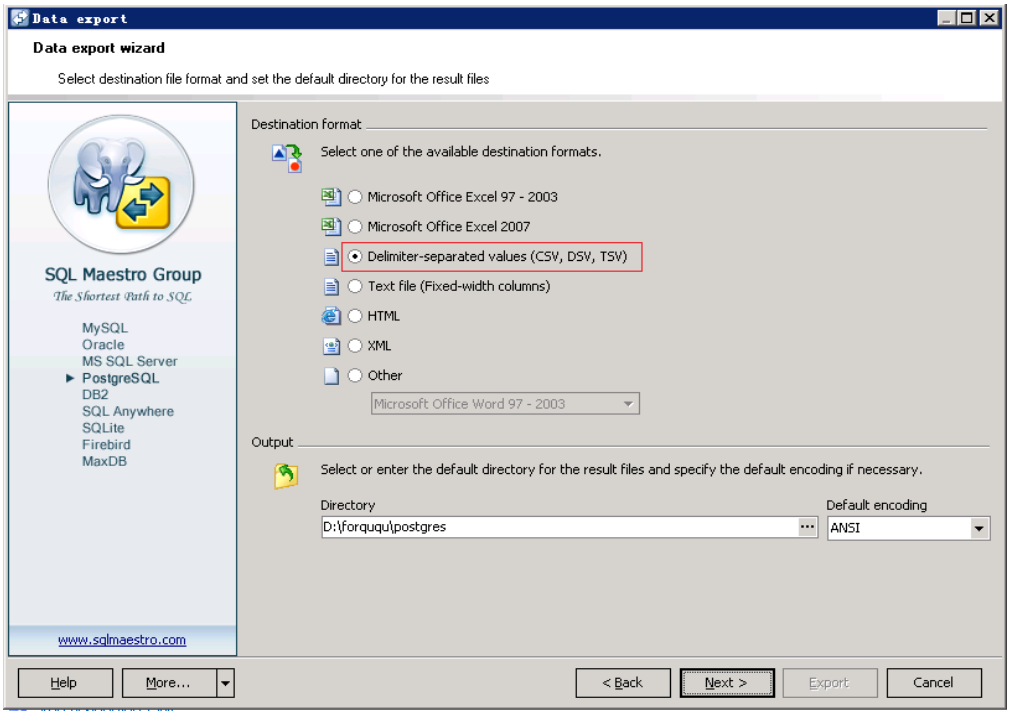


Step 2: Invoke **Data Export** by clicking on **Data Export** tool icon or hyperlink as shown in above graphic.

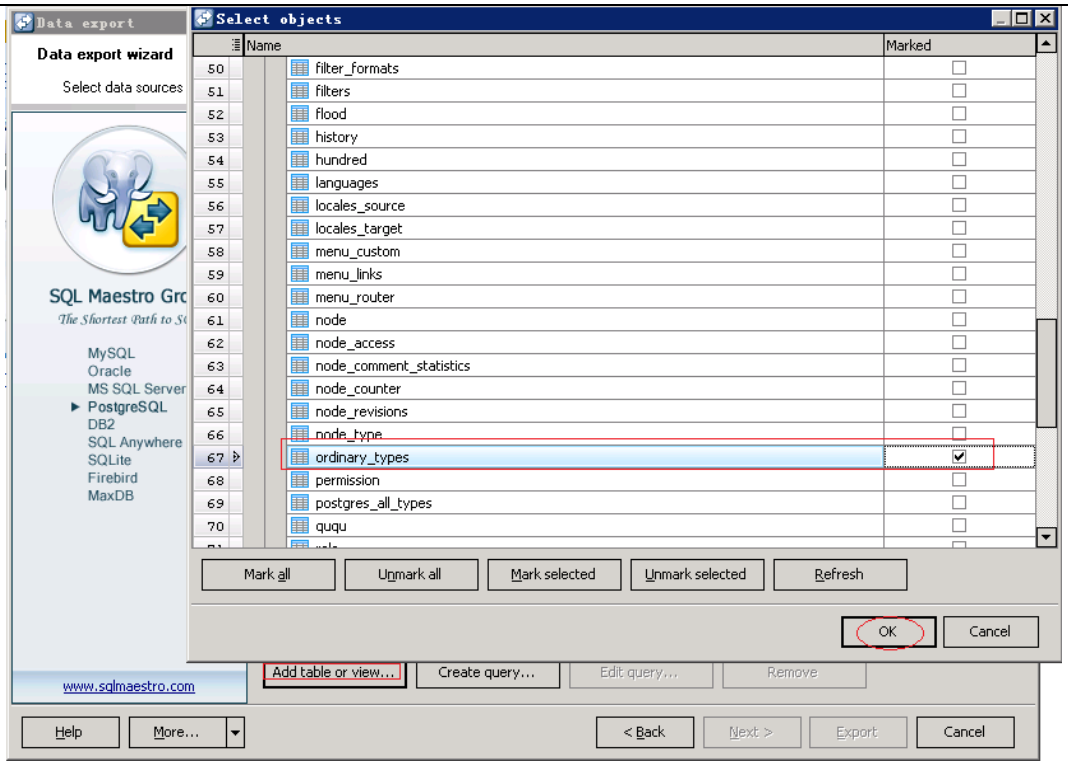


Step 3: Select CSV for output file-format

In this step, you can choose the output file format from the radio-box list; set the export file location by entering the full path or clicking on the **browse** button; choose the file encoding in the dropdown list.

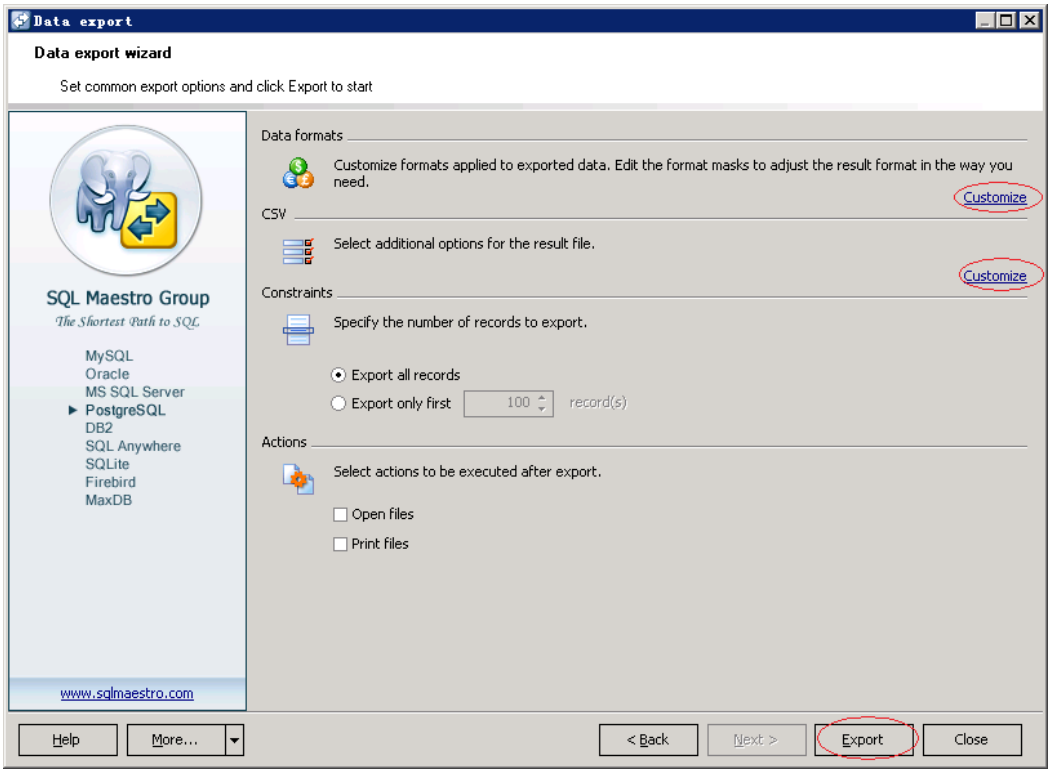


Step 4: Click on **Next** and add tables or views as following**.**



Step 5: Click on **Next** and **Next** (or **Export** directly) to generate the output file by default**.**

Of course, with **Customize** link buttons, you can define the result format for some types, and you can choose the delimiter for separating the column.



4.2.1.2 Import Data to DBMaster

Import data from appropriate format files which are exported from PostgreSQL. Currently **JDATA Transfer** supports three kinds of formats for importing: TEXT, XML and ODBC. So we should select the format both **JDATA Transfer** and **Data Wizard** can support. Here we demonstrate both

import from Text and ***import from XML*** for your reference, which can import the data exported from PostgreSQL into DBMaster.

4.2.1.2.1 Import from TEXT

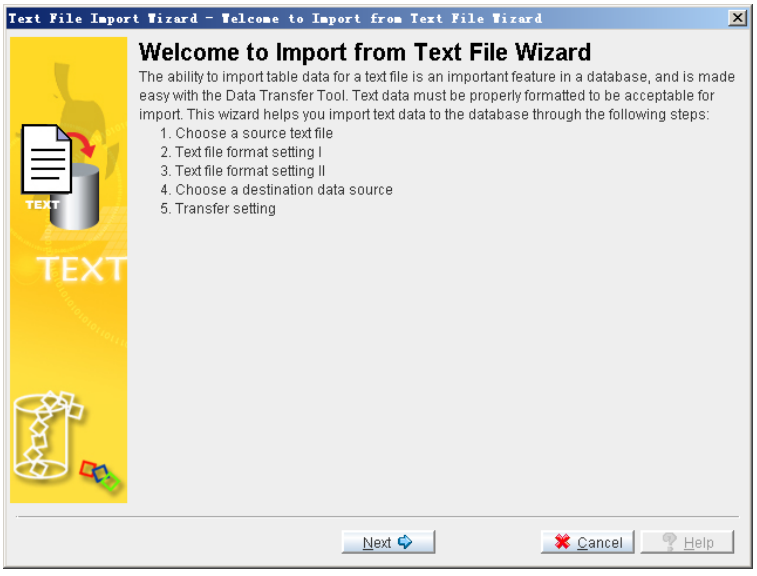
The ability to import table data from a text file is an important feature in a database, and is made easy with the **Data Transfer Tool**. Text data must be properly formatted to be acceptable for importing. Data may be imported to the database only from a properly formatted text file.

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:

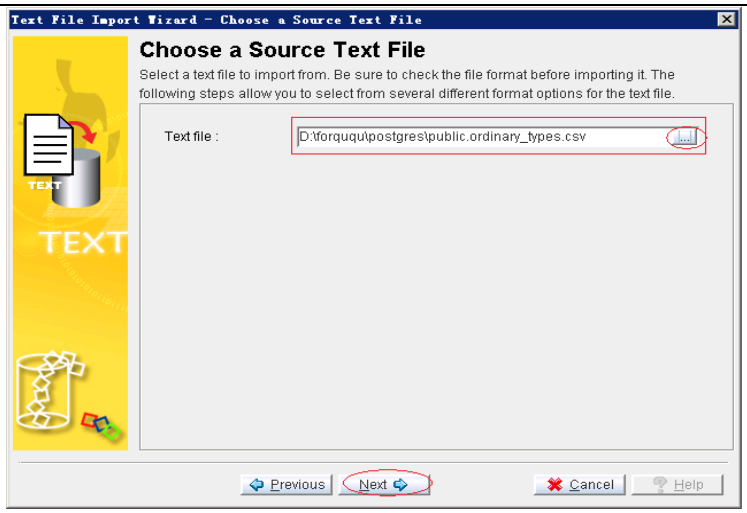
Step 1: Open the **Data Transfer Tool**.

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.

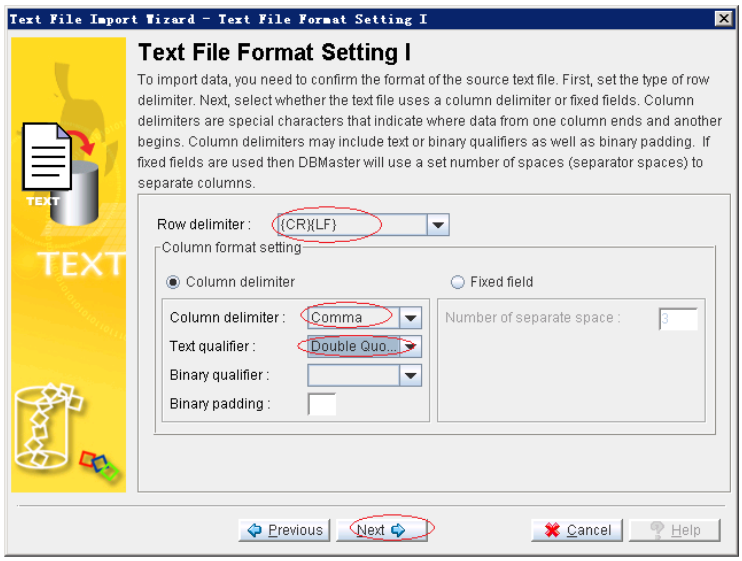


Step 3: Click on **Next,** and select a text file which exported from PostgreSQL.

You can enter the full path for the text file, or click on the **Browse** button to search for the text file.

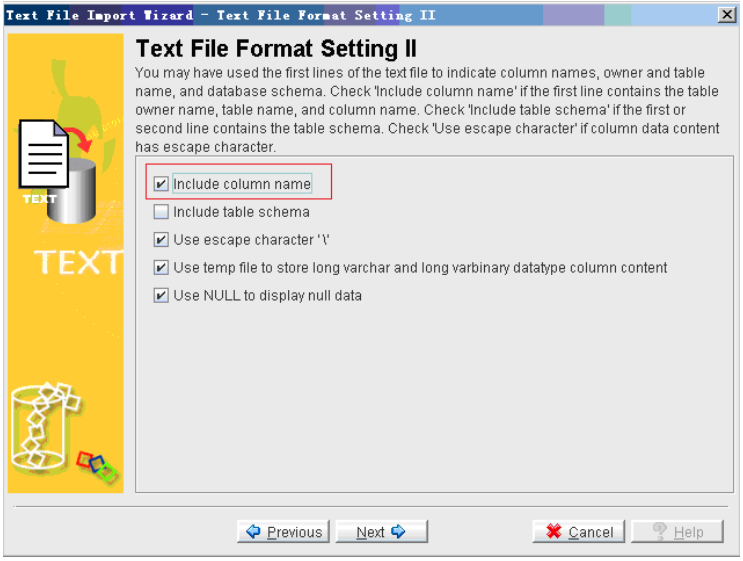


Step4: After you have selected a text file, click on **Next**, the **Text File Format Setting I** Window appears. You must choose the consistent **Column delimiter** and **Text qualifier** with the output file-format which exported from PostgreSQL.



Step 5: Click on **Next**, the **Text File Format Setting II** window appears

You should choose the **Include column name** which is consistent with exported file-format.



Line			
Lseg			
Boxes			
Paths			
Polygons			
Circles			
Network Address Types			
Inet		N/A	
Cidr			
macaddr			
Bit String Types			
bit(n) bit varying(n)	Bit strings are strings of 1's and 0's. They can be used to store or visualize bit masks. A bit string value requires 1 byte for each group of 8 bits, plus 5 or 8 bytes overhead depending on the length of the string	N/A	
New data types		New data types	
N/A		Media Types	Large object columns may also be specified as media types to aid in media process functions such as full text search for Microsoft Word documents. The following media types are available: MsWordType, HtmlType, XmlType, MsPPTType, MsExcelType, PDFType, MsWordFileType, HtmlFileType, XmlFileType, MsPPTFileType, MsExcelFileType and PDFFileType.

5.2.2 DATA TYPES MAPPING CONCERN

This section outlines conversion considerations for Datetime and LARGE OBJECT as examples to illustrate the factors you should consider:

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

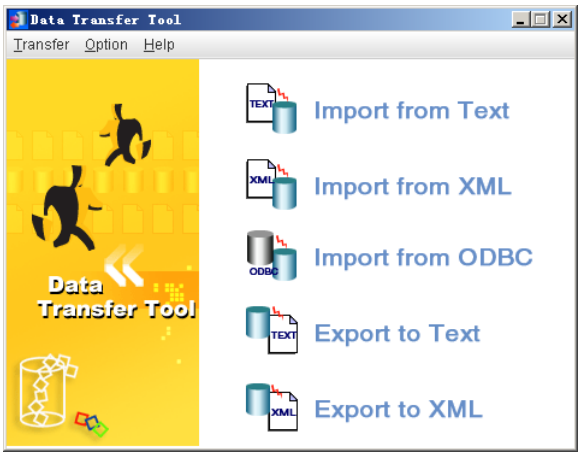
5.2.2.1 DATETIME Data Types

The timestamp/time definition and its precision in PostgreSQL differ from the same name of data types in DBMaster. For example, PostgreSQL also has the TIMESTAMP data type that stores date and time values accurate to microsecond. But the TIMESTAMP data type is only accurate to second in DBMaster. Another data types is TIME in PostgreSQL, which has a precision of microsecond. In DBMaster, TIME has a precision of 1 second. According to the description here, Migration from PostgreSQL to DBMaster will lose the precision of data in some cases.

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

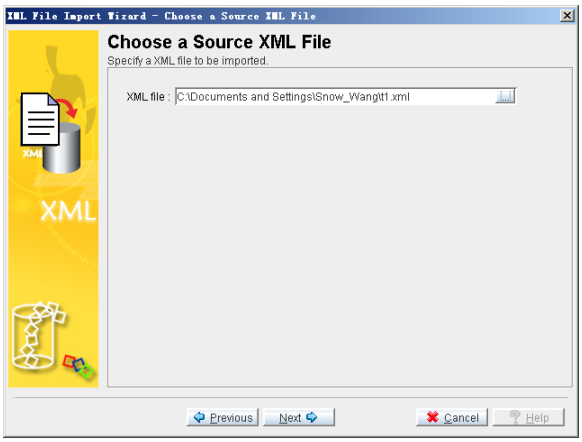
Step 1: Open the **Data Transfer Tool**.



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

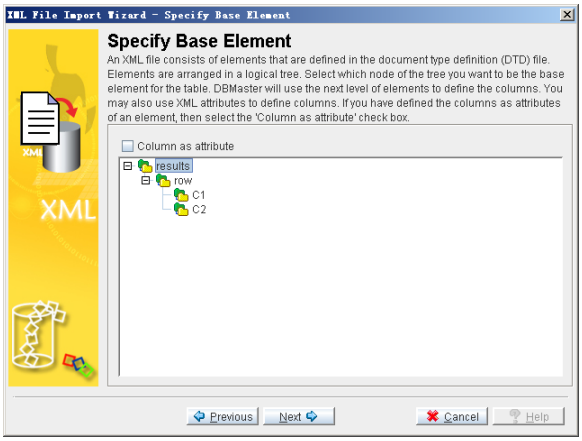
Step 3: Click on **Next** and choose a Source XML File.

You can enter the full path of a XML file to import or click the **browse** button to search for a XML file.



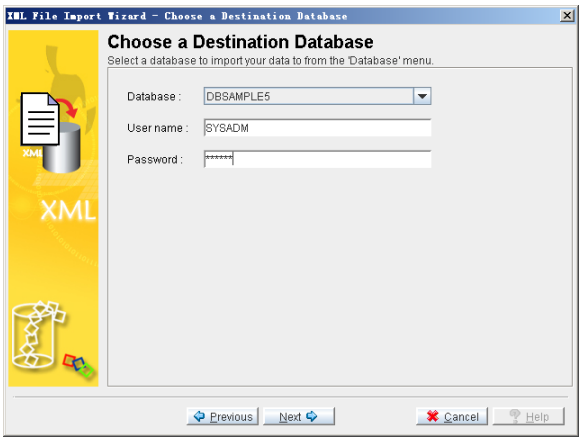
Step 4: Click on **Next**. The **Specify Base Element** window appears. The node of the tree structure represent elements in the XML file. Click on the node on the tree until they are fully expanded.

Select a parent element to be the table name. The child elements will become the columns of the table. Check out the legality of the column attribute.



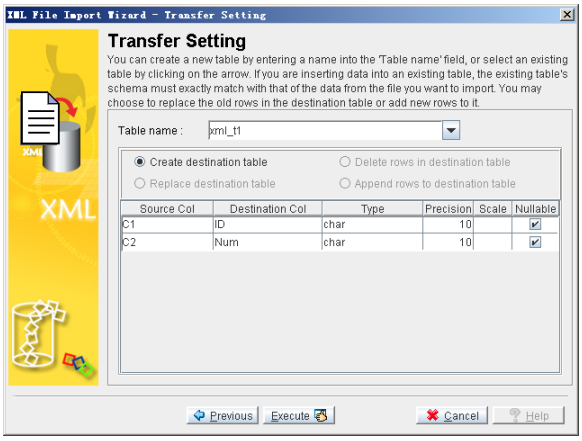
Step 5: Click on **Next** and choose the destination data source.

Select the database name from the Database menu and enter the user name and the password into the appropriate fields.



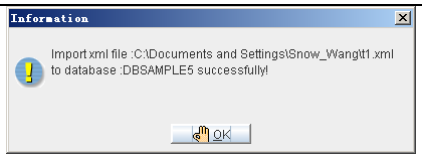
Step 6: Click on **Next.** The **Transfer Setting** window appears

You can 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 confirm dialog box appears.

Click on **OK** and check data in DBMaster.



4.2.2 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 PostgreSQL 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, and you need to ensure the script can work well on the destination database.

4.2.2.1 Migration methods

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

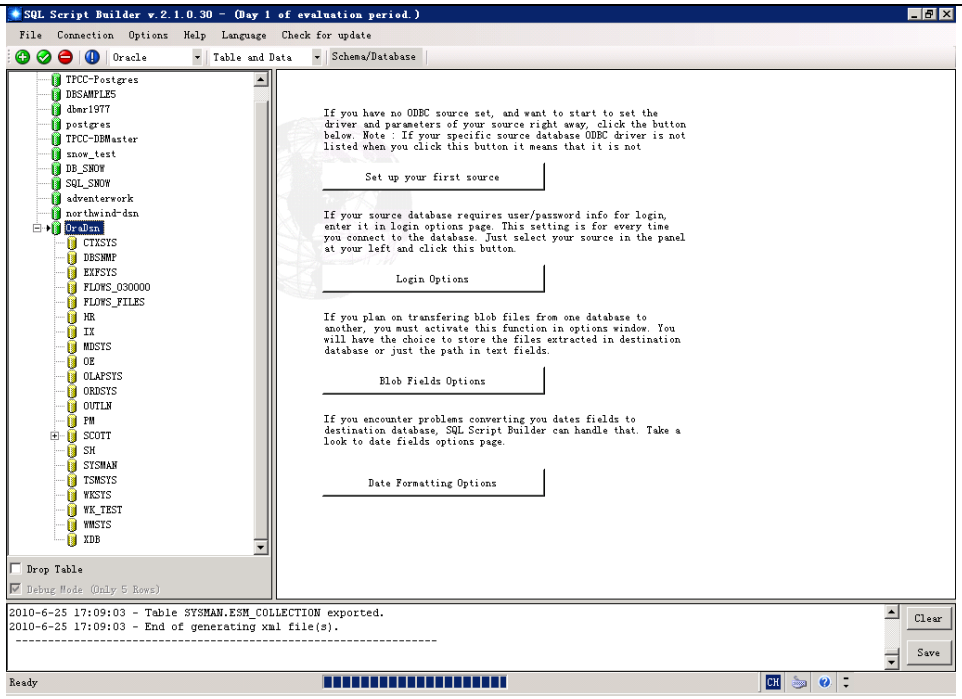
First, Use this tool to convert the data from the origination database to a supported file format. For DBMaster, we recommend the XML format.

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

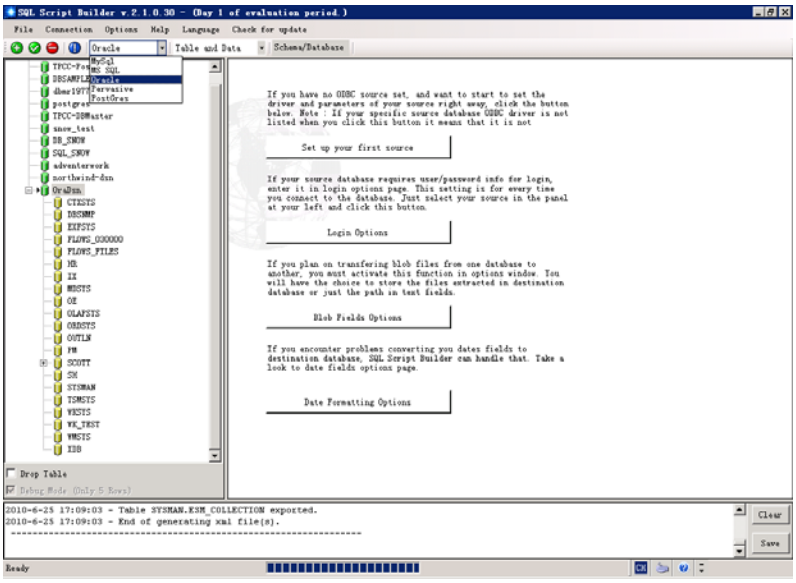
4.2.2.2 Migration steps

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

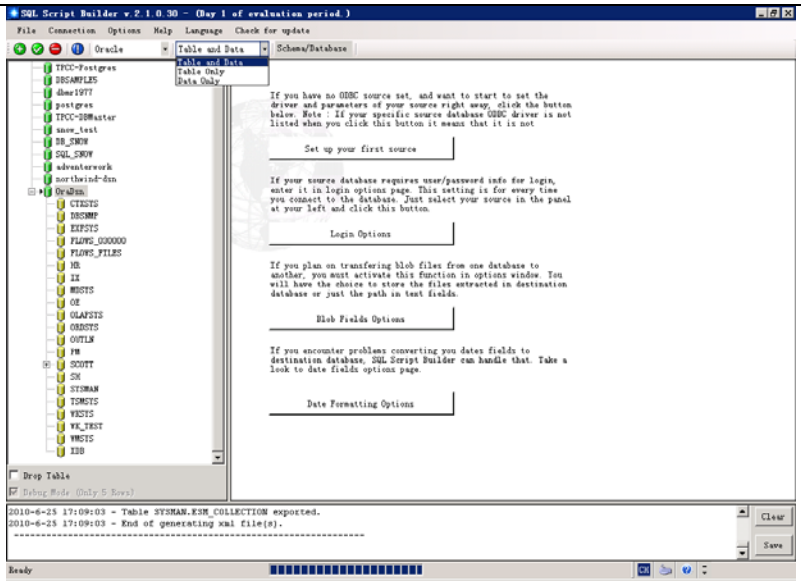
Step 1: Open **SQL Script Builder**.



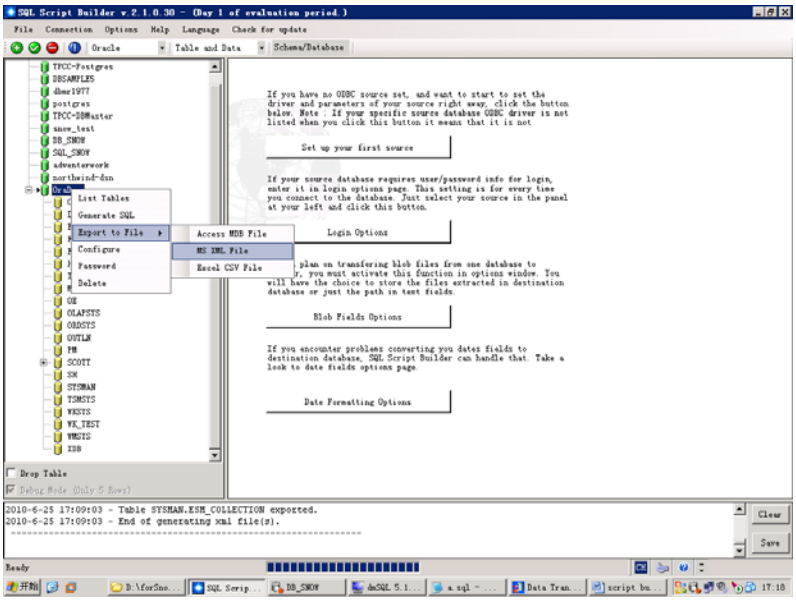
Step 2: Select the source database type from the dropdown list.



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

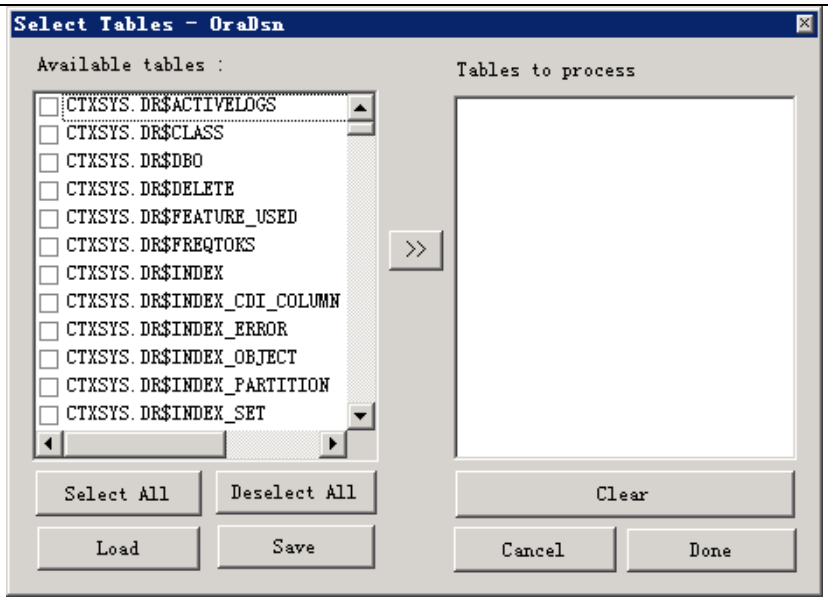


Step 4: Select XML file formats.

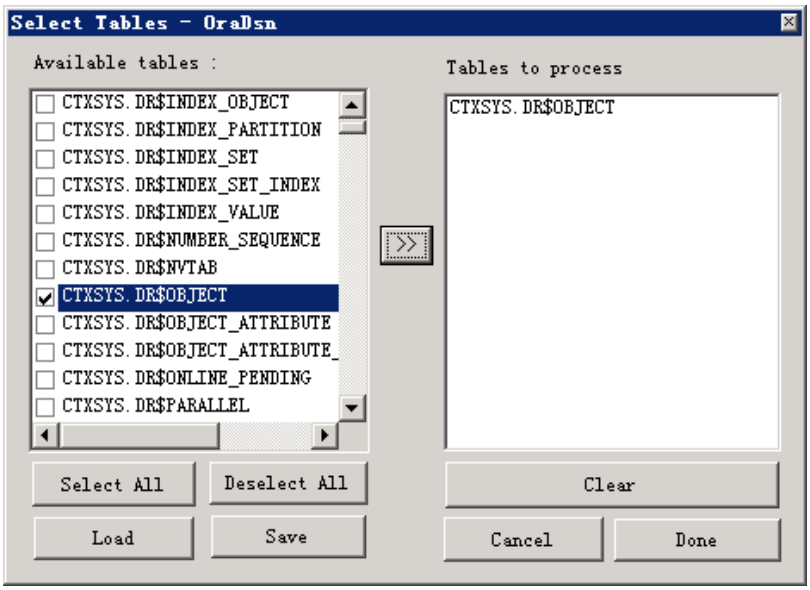


Step 5: Select export XML files location.

Step 6: Display source tables.



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



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

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

More details of importing from XML please refer to *chapter* [4.2.1.2.2](#).

4.3 Modify DDL manually

If you are familiar with DDL, you can export all schemas and data with the TXT format from PostgreSQL firstly. And modify the schema and make the syntax and data types fit to DBMaster. Much of the conversion effort can be accomplished using a simple "search-and-replace" type of approach in a text editor.

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 modify and successfully run DDL files in DBMaster. At last, export the

data from PostgreSQL with TXT or XML file format described in *chapter 4.2.1.1*. And import into DBMaster via **JDATA Transfer Tool** described in *chapter 4.2.1.2*.

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

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 PostgreSQL and DBMaster

5.1 Schema Comparison

5.1.1 THE TERMINOLOGY COMPARISON

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

PostgreSQL	DBMaster
Database	Database
Tablespace	Tablespace
Page/TOAST	Page/Frame
Role	User/ Group
Schema	Schema
Table	Table
View	View
Temporary Table	Temporary Table (temporary create '.tmp' file)
Cluster	N/A
Check constraint	Check constraint
Serial	Serial
N/A	Synonyms
Triggers	Triggers
Column default	Column default
Unique index	Unique index
Primary key	Primary key
Foreign key	Foreign key
Index	Index
PL/pgsql	Embedded-SQL (ESQL/C) stored

	procedure/ Java stored procedure / SQL SP is supported after DBMaster 5.1
UDF	UDF
Domain	Domain

5.1.2 STORAGE STRUCTURE COMPARISON

Basically, DBMaster has many differences from PostgreSQL. User should recognize these differences with discretion. Try to solicit and convert some missing setting or files to the correspondence.

Item		PostgreSQL	DBMaster
Interfaces or tools to configure parameters		N/A	JConfiguration tool
Temporary Tables		Temporarily created in a file with no extension names.	Temporarily created in '*.tmp' file
File types	Data Files	Without extension names.	*.SDB or *. DB
	Journal Files	*.LOG	*.JNL
	BLOB Files	N/A	*.SBB or *.BB

Note: a table in PostgreSQL that has columns with potentially large entries will have an associated TOAST table, which is used for out-of-line storage of field values that are too large to keep in the table rows proper.

In DBMaster, to increase database performance, there are two kinds of LOs to store large data objects: Binary Large Objects (BLOBs), which are stored in database files, and File Objects (FOs), which are stored as external files on a host file system. A BLOB, stored in database files, can only be accessed through DBMaster and insists on the data integrity provided by DBMaster, such as transaction controls, logging and recovery. A BLOB can only be shared among tuples in the same table while updating recorders. However, a FO can be shared between tables in a database. In addition, when the data needs to be shared by the other non-database applications, using FOs will be more flexible. For details, please refer to *DBA*.

5.1.3 PROCESS AND RELATED TERM DEFINITION

In PostgreSQL, every task is taken care by some specific processes. DBMaster uses the general process instead of individual processes. Therefore, a process in DBMaster usually comprises many PostgreSQL processes.

Item	PostgreSQL	DBMaster
Start-up mode	Single server supports multiple databases	Single server supports one database

Management Daemon	postmaster autovacuum	DBMaster Server Backup Server I/O server DDB server Replication server

5.1.4 RESERVED WORD CONFLICT IN DATABASE OBJECT

Reserved words vary between PostgreSQL and DBMaster. Many DBMaster reserved words are valid object names or column names in PostgreSQL, e.g ADD, AFTER, ALTER and so on. Likewise, many PostgreSQL reserved words are valid object names in DBMaster, e.g OLD, DO, FREEZE, and so on. Using 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.

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

For example (COLLATE and ***CHECK*** are both reserved words in these two database***).***

In DBMaster: create table "COLLATE"("check" int);

In PostgreSQL: create table "COLLATE"("check" int);

Different from PostgreSQL, in DBMaster, we also can set keyword **DB_ResWd** to be 0 in the 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*

PostgreSQL	DBMaster

ALL, ANALYSE, ANALYZE, AND, ANY, ARRAY, AS, ASC, ASYMMETRIC, AUTHORIZATION, BETWEEN, BINARY, BOTH, CASE, CAST, CHECK, COLLATE, COLUMN, CONSTRAINT, CREATE, CROSS, RENT_CATALOG, CRRENT_DATE, CURRENT_ROLE, CURRENT_SCHEMA, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_USER, DEFAULT, DEFERRABLE, DESC, DISTINCT, DO, ELSE, END, EXCEPT, FALSE, FETCH, FOR, FOREIGN, FREEZE, FROM, FULL, GRANT, GROUP, HAVING, ILIKE, IN, INITIALLY, INNER, INTERSECT, INTO, IS, ISNULL, JOIN, LEADING, LEFT, LIKE, LIMIT, LOCALTIME, LOCALTIMESTAMP, NATURAL, NEW, NOT, NOTNULL, NULL, OFF, OFFSET, OLD, ON, ONLY, OR, ORDER, OUTER, OVERLAPS, PLACING, PRIMARY, REFERENCES, RETURNING ,RIGHT, SELECT, SESSION_USER, SIMILAR, SOME, SYMMETRIC, TABLE, THEN, TO, TRAINLING, TRUE, UNION, UNIQUE, USER, USING, VARIADIC, VERBOSE, WHEN, WHERE,WITH,	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 contains a variety of constraints checking, data types mapping, and so on. We enlist the factors as followings.

- Entity Integrity Constraints
- Referential Integrity Constraints
- Unique Key Constraints
- Check Constraints
- Object names Limitations

5.1.5.1 Entity Integrity Constraints

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

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

PostgreSQL	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

PostgreSQL provides declarative referential integrity. A CREATE TABLE or an 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 PostgreSQL have many similarities in the term of Integrity Constraints. It makes the migration process less labor.

PostgreSQL	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, Column 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_name1 ADD FOREIGN KEY (column_name) REFERENCES table_name2 (column_name) ALTER TABLE table_name1 ADD CONSTRAINT fk_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)

FOREIGN KEY (column_name) REFERENCES table_name2(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

PostgreSQL defines unique keys as part of CREATE TABLE or ALTER TABLE statements. PostgreSQL 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 referred to as the unique index. Users should be aware of the difference between the two terminologies.

PostgreSQL	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

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

PostgreSQL	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,…);

5.1.5.5 Object name Limitation

DBMaster Objects names have a maximum length of 128 characters, and may contain numbers, spaces, letters, underscore characters and the symbols $ and #. The first character can't be numbers.

5.2 Data Types Mapping

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

5.2.1 COMMON DATA TYPE MAPPING

Specifically, this section contains the following information:

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

PostgreSQL	Description	DBMaster	Comments
Integer types		Integer types	
SMALLINT	The SMALLINT data type uses 2 bytes storage with the range of -32768 to 32767.	SMALLINT (2 byte)	Two-byte integer, 15 bits, and a sign. (-32768 to 32767)
INTEGER	The INTEGER data type uses 4 bytes storage with the range	INTEGER (4 byte)	The INTEGER data type uses 4 bytes storage with the range of –2147483648 to

	of -2147483648 to 2147483647.		2147483647.
BIGINT	The BIGINT data type uses 8 bytes storage. with the range of -9223372036854775808 to 9223372036854775807	N/A	Users should use DECIMAL instead and DBMaster supports in 5.2 version
SERIAL	The SERIAL data type uses 4 bytes storage. It is auto incrementing integers with the range of 1 to 2147483647.	SERIAL	The internal value used to generate a SERIAL number is actually an integer value; the SERIAL data type shares all of the properties of the INTEGER data type, which occupies 4 bytes of storage, The maximum value is 2,147,483,646 and a minimum value of ¨C2, 147,483,646.
BIGSERIAL	The BIGSERIAL data type uses 8 bytes storage. It is large auto incrementing integer with the range of 1 to 9223372036854775807	N/A	Users should use DECIMAL instead and DBMaster supports in 5.2 version
Floating point data types		Floating point data types	
DECIMAL/ NUMERIC	The Decimal/Numeric data type uses variable storage and user-specified precision, exacting with the range 1000.	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.
REAL	The REAL data type uses 4 bytes storage with the range of 6 decimal digits precision.	REAL	The REAL data type is an approximate signed numeric data type having a mantissa with a precision of 7. The REAL data type uses 4 bytes of storage and has a valid input range of 3.402823466E38 to –3.402823466E38. The smallest valid input values are 1.175494351E-38 and –1.175494351E-38.
DOUBLE PRECISION	The DOUBLE data type uses 8 bytes storage with the range of 15 decimal digits precision.	DOUBLE (8 byte)	The DOUBLE data type uses 8 bytes of storage and has a valid input range of 1.0E308 to –1.0E308. The smallest valid input values are 1.0E-308 and –1.0E-308.
Binary digit data types		Binary digit data types	
BYTEA	The BYTEA data type	BINARY(1-3992)	The minimum length of

	is a variable-length binary string using 1 or 4 bytes plus the actual binary string. The input format of BYTEA is different from BLOB (BINARY LARGE OBJECT), but the provided functions and operators are mostly same.	(n byte)	BINARY columns is 1 byte and the maximum length is 3992 bytes.
		BLOB(LONG VARBINARY)	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.
Boolean Type		Boolean Type	
Boolean	Boolean uses 1 byte of storage. Boolean can have one of only two states: “true” or “false”. A third state, “unknown”, is represented by the SQL null value. Valid literal values for the “true” state are: TRUE,'t', 'true', 'y', 'yes',’ on’, '1'. For the “false” state, the following values can be used: FALSE, 'f', 'false', 'n', 'no', ‘off’,'0'.	N/A	Users should use SMALLINT /CHAR(1) instead, but in DBMaster, 1 means “true”, 0 means “false”. Users’ application programs may need some modification.
Character data types		Character data types	
CHARACTER (n), CHAR (n) (n is a positive integer)	Fixed-length, blank padded. This type can store strings up to *n* characters (not bytes) in length. Characters without length specifier is equivalent to character (1). If n>32640, LONG VARCHAR should be used in DBMaster.	CHARACTER (n), CHAR (n) (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 1 character and the maximum length depends on DB_PGSIZ (4k, 8k, 16k, and 32k) (NO Unicode).
		NCHAR(n) (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 can be a minimum length of 1 character and the maximum length depends on DB_PGSIZ (4k, 8k, 16k, and 32k) (Unicode).
CHARACTER VARYING(n), VARCHAR(n)	Variable-length with limit. This type can store strings up to n characters (not bytes) in length. If character varying is used without length specifier, the type accepts strings of any size. If n>16320, LONG	VARCHAR 3968 (4KB page size) 8064 (8KB page size) 16256 (16KB page size) 32640 (32KB page size)	VARCHAR columns have a minimum length of 1 character and a maximum length depending on DB_PGSIZ (4k, 8k, 16k, and 32k). To fit DBMaster limitations, only length within 32640 bytes in PostgreSQL could use VARCHAR type in DBMaster (NO Unicode).

	VARCHAR should be used in DBMaster.	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 can be a minimum length of 1 character and the maximum length depends on DB_PGSIZ (4k, 8k, 16k, and 32k) (Unicode).
"char"	1 byte, single-byte internal types	N/A	Users should use CHAR(1) / CHAR(N) instead.
name	64 bytes, internal types for object names		
Text and image data types		BLOB data types	
TEXT	The text type variable unlimited length, Which stores strings of any length.	CLOB (LONG VARCHAR)	The maximum length of CLOB columns is 8TB.
		NCLOB	The NCLOB data type is a variable length data type that can contain any Unicode character. The maximum length for an NCLOB column is 8 TB.
BYTEA		LONG VARBINARY	Similar to the BLOB.
N/A		File	DBMaster provides the SYSTEM FO and User FO.
Date and Time types		Date and Time types	
TIMESTAMP[without time zone]	Both date and time. 8 bytes	TIMESTAMP (11 byte)	In DBMaster, the precision of TIMESTAMP is one second. Migration from PostgreSQL to DBMaster would lose the precision of data in some cases.
Timestamp[with time zone]	Both date and time with time zone. 8 bytes		
interval	Time intervals12 bytes	N/A	
Date	Date only 4 bytes	Date(4 byte)	Stored the date (accurate to day)
Time[without time zone]	Times of day only 8 bytes.	TIME (4 byte)	Migration from PostgreSQL to DBMaster would lose the precision of data in some cases.
Time[with time zone]	Times of day only, with time zone 12 bytes.	N/A	Users should use VARCHAR (12) instead.
Money data types		Money data types	
MONEY	Monetary data types, 8 bytes, values range from -92233720368547758.08 to +92233720368547758.07	N/A	Users should use DECIMAL (numeric) or INT to be the replacement or create domain replacement.
Geometric types			
Points		N/A	

Line			
Lseg			
Boxes			
Paths			
Polygons			
Circles			
Network Address Types			
Inet		N/A	
Cidr			
macaddr			
Bit String Types			
bit(n) bit varying(n)	Bit strings are strings of 1's and 0's. They can be used to store or visualize bit masks. A bit string value requires 1 byte for each group of 8 bits, plus 5 or 8 bytes overhead depending on the length of the string	N/A	
New data types		New data types	
N/A		Media Types	Large object columns may also be specified as media types to aid in media process functions such as full text search for Microsoft Word documents. The following media types are available: MsWordType, HtmlType, XmlType, MsPPTType, MsExcelType, PDFType, MsWordFileType, HtmlFileType, XmlFileType, MsPPTFileType, MsExcelFileType and PDFFileType.

5.2.2 DATA TYPES MAPPING CONCERN

This section outlines conversion considerations for Datetime and LARGE OBJECT 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 timestamp/time definition and its precision in PostgreSQL differ from the same name of data types in DBMaster. For example, PostgreSQL also has the TIMESTAMP data type that stores date and time values accurate to microsecond. But the TIMESTAMP data type is only accurate to second in DBMaster. Another data types is TIME in PostgreSQL, which has a precision of microsecond. In DBMaster, TIME has a precision of 1 second. According to the description here, Migration from PostgreSQL to DBMaster will lose the precision of data in some cases.

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 PostgreSQL. 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 a FILE data type. DBMaster uses FULL PATH link to indicate the 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 PostgreSQL, BINARY data type is bytea. The SQL standard defines a different binary string type, called BLOB or BINARY LARGE OBJECT. The input format is different from bytea, but the provided functions and operators are mostly same.

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.

PostgreSQL	Description	DBMaster	Comments

B*-tree indexes	B*-tree indexs is the standard type of indexs available in PostgreSQL, and it's very useful for selecting rows that meet an equivalence criterion or a range criterion.By default, the CREATE INDEX command creates B-tree indexes in PostgreSQL B-trees can handle equality and range queries on data that can be sorted into some ordering. In particular, the PostgreSQL query planner will consider using a B-tree index whenever an indexed column is involved in a comparison using one of these operators: < ,<= ,= ,>= ,> Constructs equivalent to combinations of these operators, such as BETWEEN and IN, can also be implemented with a B-tree index search. 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 creating 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]
Hash indexes	Hash indexes can only handle simple equality comparisons. The query planner will consider using a hash index whenever an indexed column is involved in a comparison using the = operator. (Hash indexes do not support IS NULL searches.) For example: a Hash index CREATE INDEX idx_name ON table_name USING HASH (column);	N/A	
GiST indexes	GiST indexes are not a single kind of index, but rather an infrastructure within which many different indexing strategies can be implemented.	N/A	

GIN indexes	GIN indexes are inverted indexes which can handle values that contain more than one key, arrays for example. Like GiST, GIN can support many different user-defined indexing strategies and the particular operators with which a GIN index can be used vary depending on the indexing strategy.	N/A	

N/A		**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, text indexes are created on column by using Order By clause. Rebuild the text index if you load data after creating text indexes. Text index names must be unique to 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 PostgreSQL and DBMaster. The following topics are present in this section:

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

- Comparison Operators
- Arithmetic Operators
- String Operators
- Set Operators
- Bit Operators
- Built-In Functions
- String Functions
- Date Functions
- Mathematical Functions
- Locking Concepts and Data Concurrency Issues
- Locking
- Row-Level Versus Page-Level Locking
- Read Consistency
- Logical Transaction Handling
- Trigger
- Stored Procedure and Stored Function
- User-defined types
- Privileges

5.4.1 CONNECTING TO THE DATABASE

PostgreSQL	DBMaster	Description
\connect (or \c) [dbname [username] [host] [port]]	Connect to DB_NAME USER_NAME PASSWORD;	

Recommendations:

PostgreSQL can support multiple databases in one Server, and for DBMaster, users can only start one database on one server.

5.4.2 SELECT STATEMENTS

PostgreSQL	DBMaster	Description
Select clause	Select clause	
SELECT [ALL | DISTINCT]{select_list} { [LIMIT { number | ALL }] [OFFSET number] }	SELECT [ALL | DISTINCT] {select_list} {LIMIT offset, count}	The ALL keyword means every record regardless of its duplicate occurrence. However, using DISTINCT keyword will eliminate the duplicate rows. Users can extract the count rows from the offset row by using “LIMIT offset, count” both in DBMaster and PostgreSQL.
From clause	From clause	

SELECT … [FROM *table* [*alias*] [, ...]]	SELECT … FROM [[db-name[@server-name]:]user.]table-name|view] [ALIAS]	The main difference is DBMaster must use DB_NAME@HOST:USERNAME. TABLE_NAME as its full valid table name.
Join table	Join table	
T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN *T2* ON *boolean_expression*	table-reference [,] { LEFT OUTER JOIN | LEFT JOIN | OUTER } { table-reference | (table-reference-list) }	Note that PostgreSQL provides "FULL OUTER JOIN", which is not implemented in DBMaster. Users should use workaround to generate the wanted results.
T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN *T2* USING (*join column list*)		
T1 NATURAL { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN *T2*		
Where,Group by ,having ,into clause	Where,Group by, having ,into Clause	
SELECT… INTO [*new_table*] [FROM] … [WHERE condition] [GROUP BY column [, ...]] [HAVING condition [, ...]] [UNION [ALL] select] [ORDER BY column [ASC | DESC] [, ...]]	SELECT …FROM …[WHERE] [GROUP BY (column-identifier [, column-identifier]…)] [HAVING search-condition] [ORDER BY column-identifier [ASC | DESC] [INTO] [table_identifier] INSERT INTO <table> SELECT FROM....	Although both DBMRS support the INTO CLAUSE, it lies on different position in the statement.

5.4.3 INSERT STATEMENTS

PostgreSQL	DBMaster	Description
INSERT INTO *table* [(*column* [, ...])] { VALUES (*expression* [, ...]) | SELECT *query* }	INSERT INTO remote-table-name [(column-identifier [, column-identifier]...)] { VALUES (insert-value[,insert-value]...) | DEFAULT VALUES | select-order-by-statement }	Inserting can only be done on single table.

Recommendations:

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

5.4.4 UPDATE STATEMENTS

PostgreSQL	DBMaster	Description

UPDATE table SET column = expression [, ...] [WHERE CURRENT OF cursor-name] | [WHERE search-condition]	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) 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.

5.4.5 DELETE STATEMENTS

PostgreSQL	DBMaster	Description
DELETE FROM table WHERE condition] | [WHERE CURRENT OF cursor-name]	DELETE FROM remote- table-name [table-option] [WHERE condition] | [WHERE CURRENT OF cursor-name]	

5.4.6 OPERATORS

5.4.6.1 Operator comparison

Operator	Same in both Databases	PostgreSQL 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 and less than or equal to y	BETWEEN x AND y		
Pattern Matches	LIKE	ILIKE (~~*), Similar to NOT ILIKE (!~*)	Contain, Match, contains
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/SOME, <> ANY/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, NOT IN		
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	||		
Distinct row from either query	UNION		
All rows from both queries	UNION ALL		

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 are applied to a text index search. You can see the difference in the following case: DBMaster supports “Contain” and “Match” for pattern search.

Table: Dept	
ID	Name
01	DBMaster Support

```
SQL1: select * from Dept where Name like ‘DBMaster S%’(Scan the whole record, 1 result)
SQL2: select * from Dept where Name contain ‘DBM’ (Scan the string with ‘DBM’ letters. Return 1 result)
SQL3: select * from Dept where Name match ‘DBMaster’ (Scan the precise word, 1 result)
SQL4: select * from Dept where Name match ‘DBMaster S’ (Scan the precise word, no result)
```

5.4.6.3 Special operators recommendation

- No “Intersect” or “Except” operators supported

Both DBMaster and PostgreSQL provide UNION to compose the results to the whole, but only PostgreSQL support INTERSECT keyword. PostgreSQL provides three methods to handle set of results, i.e., UNION, INTERSECT and EXCEPT. Currently, DBMaster only supports “UNION” operators. As to the “Intersect” and “EXCEPT”, 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 can use “MATCH” operator to seek the full word.

- PostgreSQL supports “SIMILAR TO” and “substring function” for pattern search

There are three separate approaches to pattern matching provided by PostgreSQL: the traditional SQL LIKE operator, the more recent SIMILAR TO operator), and POSIX-style regular expressions. Functions are available to extract or replace matching substrings and split a string at matching locations. The SIMILAR TO operator succeeds only if its pattern matches the entire string, like LIKE, SIMILAR TO uses _ and % as wildcard characters denoting single characters and strings.

The substring function with three parameters, substring (string, from pattern, for escape-character), provides extraction of a substring that matches an SQL regular expression pattern. As with SIMILAR TO, the specified pattern must match the entire data string, or else the function fails and returns null. To indicate the part of the pattern that should be returned on success, the pattern must contain two occurrences of the escape character followed by a double quote ("). The text matching the portion of the pattern between these markers is returned.

5.4.7 BUILT-IN FUNCTIONS

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

- Math/Number Functions
- String Functions
- Conversion Functions
- Date Functions

It doesn't include all PostgreSQL functions. For example, PostgreSQL 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 PostgreSQL are not put here for their uniqueness.

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

5.4.7.1 Math/Number Functions:

The numeric function performs 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.

PostgreSQL	DBMaster	Description
abs(n)	ABS(n)	Return the absolute value of n as a double-precision floating-point number. The return type is same as the input type.
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 arc sin 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(y, x)	ATAN2(x,y)	Return the arc tangent of x/y in the range -pi to pi as a double precision floating-point number.
ceiling(dp or numeric)	CEILING(n)	Return the least integral value greater than or equal to n as a double-precision floating-point number. The return type is same as the input type.
cos(n)	COS(n)	Return the cosine of n as a double-precision floating-point number. n is expressed in radians.
Cot(n)	COT(n)	return the cotangent of *number,* expressed in radians
N/A	COSH(n)	Return the hyperbolic cosine of *x*.

exp(n)	EXP(n)	Return the exponential function e**n. The return type is same as the input type.
floor(n)	FLOOR(n)	Return the greatest integral value less than or equal to n. The return type is same as the input type.
ln(n)	LOG(n)	Return the natural logarithm of n. The return type is same as the input type.
log(n)	LOG10(n)	In DBMaster, return the logarithm to base 10. The return type is same as the input type.
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)	Return m**n as a double-precision floating-point number.
ROUND(n[,m])	ROUND(n[,m])	Return the closest integer number of the real number n.
sign(n)	SIGN(n)	Return the sign of a number codes as +1 for positive, 0 for zero, and -1 for negative.
sin(n)	SIN(n)	Return the sine of n as a double-precision floating-point number. N is expressed in radians.
N/A	SINH(n)	Return the hyperbolic sine of *x*
sqrt(n)	SQRT(n)	Return a double-precision floating-point number y where y*y = n. The return type is same as input type.
tan(n)	TAN(n)	Return the tangent of n as a double-precision floating-point number. N is expressed in radians.
N/A	TANH(n)	Return the hyperbolic tangent of *x*.
Degrees(n)	DEGREES(n)	Return the number of degrees in radians as a double precision floating-point number
Radians(n)	RADIANS(n)	Return the number of radians in degrees as a double precision floating-point number.
pi()	PI()	Return 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 String Functions:

Character functions accept characters 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.

PostgreSQL	**DBMaster**	**Description**
ASCII(string)	ASCII(string)	Return the ASCII code value of the leftmost character of string as an integer.
CHR(code)	CHAR(code)	Convert the decimal code for an ASCII character to the corresponding character. These 2 functions between DBMaster and PostgreSQL are identical.
Position(textsrc in textdst);	LOCATE(textsrc, textdst, start)	Locate a string in another string. But the function in DBMaster can specify the start position, and the one in PostgreSQL can not.
LENGTH(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 PostgreSQL are identical.
SUBSTR (string, start, length)	SUBSTRING(string, start, length)	Return the part of the string.
COALESCE(expression [,…,n])	COALESCE (variable, new_value)	If the value of the variable is NULL, the new_value is returned.

CASE WHEN condition THEN result [WHEN ...] [ELSE result] END	CASE WHEN exp1 THEN result1 WHEN exp2 THEN result2 ELSE default_value END	CASE compares expr to each search value one by one. If expr is equal to a search, then returns the corresponding result.
REPEAT(string_exp, count)	REPEAT(string_exp, count)	Produce a string with string_exp repeated 'count' times.
UPPER(string)	UPPER(String), UCASE(String)	Convert lowercase characters to uppercase characters. These 3 functions between DBMaster and PostgreSQL are identical.
LOWER(string)	LOWER(string), LCASE(string)	Convert all upper case characters in string to lower case. These 3 functions between DBMaster and PostgreSQL are identical.
LTRIM(string)	LTRIM(string)	Truncate trailing spaces from the left end of string. These 2 functions between DBMaster and PostgreSQL are identical.
RTRIM(string)	RTRIM(string)	Truncate the trailing spaces from the right end of string. These 2 functions between DBMaster and PostgreSQL are identical.
TRIM(string)	TRIM(string)	Truncate the trailing spaces from both end of string.
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
REPLACE(string1, string2, string3)	REPLACE(string1, string2, string3)	Replace all occurrences of string2 in string1 with string3. These 2 functions between DBMaster and PostgreSQL are identical.
N/A	CONCAT(string1, string2) ||	Return a character string that is the result of concatenating string2 to string1. The resulting string is DBMS dependent.

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.

PostgreSQL	DBMaster	Description
CAST(expression as Data type) convert(string text, [src_encoding name,] dest_encoding name)	CAST(Column as Data type)	DBMaster and PostgreSQL use CAST function to cast one data type to another.

5.4.7.4 Date Functions

PostgreSQL	DBMaster	Description
N/A	ADD_DAYS(DATE date_val,INT s)	Add the number of days to the date contained in date_val.
age(timestamp, timestamp)	DAYS_BETWEEN(DATE date1,DATE date2)	Return the number of days between the given two dates in DBMaster. But in PostgreSQL it returns as a format consisting of the year, month and day.
N/A	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	CURDATE()	Return current date.
NOW()	NOW()	Return current date and time as a timestamp value.

DATE_PART(TEXT, TIMESTAMP)	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.
N/A	LAST_DAY(dateval)	Return the last date of the month which dateval belongs to.
N/A	NEXT_DAY(dateval,weekday)	Return the date of the next first WeekDay.

5.4.8 LOCKING CONCEPTS AND DATA CONCURRENCY ISSUES

PostgreSQL	DBMaster
➢ PostgreSQL supports table-lock and row-lock.	➢ DBMaster supports the row-lock, page-lock and table-lock. ➢ DBMaster supports the “Dirty Read”, “Read with shared lock”, and “Read with exclusive lock”

Recommendations:

DBMaster, uses the “select * for update” to prevent other session from updating the locked data. Basically, the reader of the data is never locked both in PostgreSQL and DBMaster. But users should be aware of the different manners when a “select” command is submitted on these two databases and their consequences.

PostgreSQL	DBMaster
➢ PostgreSQL actually treats every SQL statement as being executed within a transaction. If you do not issue a BEGIN command, and then each individual statement has an implicit BEGIN and (if successful) COMMIT wrapped around it. ➢ COMMIT 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: ➢ ROLLBACK savepoint_name ➢ Two-phase commit allows transactions to be "prepared" on several computers, and once all computers have successfully prepared their transactions (none failed), all transactions can be committed.	➢ DBMaster transactions are explicit. ➢ Statements are automatically committed to the database by default. But users 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 and PostgreSQL. Therefore, applications expect that every statement they issue is automatically committed after it is executed. In DBMaster, you could use “DB_ATCMT=1” in dmconfig.ini to change this manner and In PostgreSQL you can use “begin” before starting a transaction to start a transaction until ‘commit’ to end a transaction.

5.4.9 TRIGGER DIFFERENCE

PostgreSQL	DBMaster
CREATE TRIGGER name { BEFORE | AFTER }{ event [OR ...] }ON table FOR EACH { ROW | STATEMENT } EXECUTE PROCEDURE funcname (arguments)	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 name ON table RENAME TO newname	ALTER TRIGGER trigger_name REPLACE WITH
DROP TRIGGER *name* ON *table*	DROP TRIGGER Trigger_name FROM Table_name

Recommendations:

There are much differences in using triggers between PostgreSQL and DBMaster. Before creating a trigger in PostgreSQL, the trigger action must be defined in a function. But in DBMaster, we needn't do like this, because we defined the trigger action when creating a trigger.

5.4.10 STORED PROCEDURES AND STORED FUNCTIONS

PostgreSQL	DBMaster
CREATE [OR REPLACE] FUNCTION *name* ([[*argmode*] [*argname*] *argtype* [, ...]]) [RETURNS *rettype*] { LANGUAGE *langname* | IMMUTABLE | STABLE | VOLATILE | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT | [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER | AS '*definition*' | AS '*obj_file*', '*link_symbol*' } ... [WITH (*attribute* [, ...])]	CREATE PROCEDURE procedure-name [(procedure-parameter [, procedure-parameter ...])] { [RETURNS STATUS] | [RETURNS [STATUS,] procedure-result [,procedure-result ...]] } | CREATE PROCEDURE FROM source-file-path

Recommendations:

There are currently four procedural languages available in the standard PostgreSQL distribution: PL/pgSQL, PL/Tcl, PL/Perl, and PL/Python. Other languages can be defined by users. We can use these languages to write functions and store procedures. On the other hand, DBMaster uses the ESQL/C for ESQL/C stored procedure or Java for Java stored procedure to do the coding. This is the biggest difference between these two databases.

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 …", Coding in Procedural Language in DBMaster is out of scope in this document. For details, please refer to the *ESQL C Programmer's Guide*.

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

5.4.11 USER-DEFINED TYPES

PostgreSQL	DBMaster
CREATE TYPE name AS (attribute_name data_type [, ...]) CREATE TYPE name (INPUT = input_function, OUTPUT = output_function [, RECEIVE = receive_function] [, SEND = send_function] [, ANALYZE = analyze_function] [, INTERNALLENGTH = { internallength | VARIABLE }] [, PASSEDBYVALUE] [, ALIGNMENT = alignment] [, STORAGE = storage] [, DEFAULT = default] [, ELEMENT = element] [, DELIMITER = delimiter])	CREATE DOMAIN domain_name as data_type [default {literal | constant | function_name | NULL}] [CONSTRAINT constraint_name CHECK Boolean_expression]

There are much differences in creating user_defined types between PostgreSQL and DBMaster.

In PostgreSQL, there are two types of CREATE TYPE to create user_defined types. The first form of CREATE TYPE creates a composite type. The composite type is specified by a list of attribute names and data types. This essentially the same as the row type of a table, but using CREATE TYPE avoid the need to create an actual table when all that is wanted is to define a type, A stand-alone composite type is useful as the argument or return type of a function. The second form of CREATE TYPE creates a new base type. The parameters may appear in any order, not only that illustrated above, and most are optional. Two or more functions (using CREATE FUNCTION) should be registered before defining the type. The support functions input_function and output_function are required, while the functions receive_function, send_function and analyze_function are optional.

In DBMaster, we use command ‘create domain’ to define a user-defined type. A domain is a type of integrity constraint used to defining a column. Domains specify the data type for the column, and may specify a default value or a value constraint. When a column is defined with a domain, it inherits the properties of the domain, (data type, default value, and value constraint), without requiring the user to specify them.

5.4.12 PRIVILEGES

PostgreSQL	DBMastker	Description
SELECT	SELECT	Allow select from any column of the specified table, view,or sequence.
INSERT	INSERT	Allow insert of a new row into the specified table.
UPDATE	UPDATE	Allow update of any column of the specified table.
DELETE	DELETE	Allow delete of rows from the specified table.
TRUNCATE	N/A	Allow TRUNCATE on the specified table in PostgreSQL.
REFERENCES	REFERENCE	To create a foreign key constraint, it is necessary to have this privilege on both the referencing and referenced tables.
EXECUTE	EXECUTE	Allow the use of the specified function and any operators that are implemented on top of the function in PostgreSQL. AND Allow the use of the COMMAND, PROJECT, and PROCEDURE in DBMaster.

N/A	ALTER	Allow users to alter the definition of a table in DBMaster.
CREATE	N/A	Allow new schemas to be created within the database in PostgreSQL.
CONNECT	CONNECT	Allow the user to connect to the specified database.
N/A	INDEX	Allow users to create or drop indexes for a table in DBMaster.
RULE	N/A	Allow the creation of a rule on the table/view in PostgreSQL.
TRIGGER	N/A	Allow the creation of a trigger on the specified table in PostgreSQL.
TEMPORARY	N/A	Allow temporary tables to be created when using the database in PostgreSQL.
USAGE	N/A	In PostgreSQL, allow the use of the specified language for the creation of functions in that language. This is the only type of privilege that is applicable to procedural languages.

5.4.13 POSTGRSQL AND DBMASTER IN AP

PostgreSQL	**DBMaster**
Supported driver	
JDBC/ODBC,Perl DBI,Hibernate,Nhibernate,OLE DB,.NET	JDBC/ODBC,DCI, Ruby, Hibernate, Nhibernate,OLE DB
Connection String	
ODBC: Driver={PostgreSQL}; Server=IP Address; Port =5432; Database=DataBase; Uid=Username; Pwd=Password;"	ODBC: "Driver={DBMaster 5.1 Driver};Database=Database name;uid=Username;Pwd=Password;"
OLE DB: “Provider = PostgreSQL OLE DB Provider; Data Source=LocalHost; User ID=Username; Password=; Location=Database; Extended Properties=”	OLE DB: "Provider=DMOLE51;Data Source= Databasename;User Id= Username; Password =;"
JDBC: Class.forName (“org.postgresql.Driver ”); Connection conn= DriverManager.getConnection ("jdbc:postgresql://host:port/database","user","password") ;	JDBC: Class.forName("dbmaster.sql.JdbcOdbcDrive r "); 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:

PostgreSQL	**DBMaster**
Check one table exist	
select 1 from pg_tables where tablename= ‘XXXXX’;	select 1 from systable where table_name=’ XXXXX’;
Check DB Version from SQL	
select substring(version() from 12 for 5);	select value from sysinfo where info='VERSION';

Check Procedure exist	
select count(*) from pg_proc where proname =' XXXXXXX';	SELECT COUNT(*) FROM sysprocinfo WHERE NAME = '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 PostgreSQL to DBMaster.

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

6.2 CONVERT USER-DEFINED TYPES

There are huge differences between PostgreSQL and DBMaster on User-defined types. Please read the detailed introduction about it in *chapter 5 sections 5.4.11*

First, we should analyze the User-defined types function in PostgreSQL.

Second, we can rewrite “domain” according to the syntax of DBMaster.

Note: Please spend some time on a careful technical evaluation before using it.

6.3 CONVERT TRIGGER

PostgreSQL is similar to DBMaster in many aspects, both of them only include DML Triggers. This article will focus on (DML) triggers.

Difference in the Trigger between PostgreSQL and DBMaster has been introduced in *chapter 5 section 5.4.9*.

First, we should analyze the PostgreSQL Trigger.

Second, we can rewrite trigger according DBMaster syntax.

6.4 CONVERT STORED PROCEDURE

Detailed Recommendations for stored procedures between PostgreSQL and DBMaster have been introduced in *chapter 5.4.10*. **Stored Procedures and Stored Functions**. Here we mainly discuss how to convert stored produces from PostgreSQL to DBMaster successfully. PostgreSQL stored procedures use the PL/SQL but DBMaster use 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.

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

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

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 from another database.

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

Finally, mark the special syntax in PostgreSQL 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 providers or divers are used to access data source. JDBC, ODBC or any others, for example: If data providers changes, we might consider changing drivers.

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 related to database connection or manipulation may need some modifying. Such as DSN, CONNECT SRTING, and so on on client.

In addition, if the application need to 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 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 existing on server separately and whether these programs need modifying.

7.2 PostgreSQL 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 PostgreSQL. Consider what method is substitute for these special grammars.

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

7.2.1 FOR SELECT STATEMENT

In PostgreSQL, select statements don't need parentheses in the function end sometimes. For example, you should write "select current_date;" to query time; otherwise, an error will return. but in DBMaster, you must write "select curdate ();".

7.2.2 FOR INHERITANCE AND PARTITIONING

PostgreSQL implements table inheritance, In PostgreSQL, a table can inherit from zero or more other tables, and a query can reference either all rows of a table or all rows of a table plus all of its descendant tables. The latter behavior is default.

PostgreSQL also supports partitioning via table inheritance.

Currenty, DBMaster doesn't support these function temporarily.

7.2.3 FOR NESTED QUERY

Suppose we have a table named tb_nest recording all staff information .If we want to know who is latest for each department.

In PostgreSQL, 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 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 on, 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 had no problems. This is very helpful for you to migrate all applications programs from PostgreSQL to DBMaster successfully.

8.2 Test application with DBMaster after migration

A validate testing is required after application programs have been migrated completely from PostgreSQL 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 after migration .The amount data is growing in database. You should often pay attention to database performance tuning. If databases' performance gets down, we should detect database and adjust timely in use.

Performance tuning is not only until migration finished from PostgreSQL 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 and developing an application with better utilities both can 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 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 applications' throughput. In addition, shortening transactions can promote concurrency, Meanwhile, it is also possible to degrade databases' performance.

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 page buffers, journal buffers and system control area.

To achieve the best performance, follow the following steps:

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.

The 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 smoothly and efficiently.

Memory swapping between the physical memory and 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 utility. An extremely high page-swapping rate indicates that the amount of physical memory in a system is not large enough.

In this case, you should remove 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 server. When 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 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 appropriate parameters **DB_NBufs**, **DB_NJnlB** and **DB_ScaSz** in **dmconfig.ini** before starting the database to configure the size of the DCCA components.

The total memory allocation for the DCCA is 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 in 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 measure the number of used journal blocks and the other is measure 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 when 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 performance of the whole system, so please chose one special designed OS 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 in order to improve the performance of database application programs. 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 PostgreSQL. The content involves samples for not only table schemas and data but also applications with different program languages. It provides a good demonstration of Migration from PostgreSQL to DBMaster.

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

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

10.1 Table Schema for all Types

In order to make users to get to know types mapping between PostgreSQL and DBMaster, we give an example here. Users should create a table with all types in PostgreSQL firstly. Then, modify the table schema according to the Type-Mapping table for creating table in DBMaster manually, or export the table from PostgreSQL to DBMaster with **JDatatransfer Tool** automatically.

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

10.1.1 CREATE TABLE WITH ALL TYPES IN POSTGRESQL

Some types are equivalent to other types, or they only have the various aliases. And they will be converted into some same types automatically when creating the table. For example:

TYPE Name	Equivalent	Alias
Smallint		int2
int, int4, serial, serial4	integer	
bigint, bigserial, serial8	int8	
float, real		float4
double precision		float8
numeric		decimal
bit varying		varbit
boolean		bool
character		char
character varying		varchar

```
create table postgres_all_types(
```

```
col_smallint          int2,
col_integer           int,
col_bigint            int8,
col_serial            serial4,
col_bigserial         serial8,
col_numeric           decimal(13,3),
col_real              float4,
col_doubleprecision   float8,
col_bit               bit(20),
col_bitvarying        varbit(20),
col_boolean           bool,
col_character         char(30),
col_charactervarying  varchar(30),
col_text              text,
col_bytea             bytea,
col_cidr              cidr,
col_inet              inet,
col_interval          interval,
col_macaddr           macaddr,
col_money             money,
col_point             point,
col_line              line,
col_lseg              lseg,
col_box               box,
col_path              path,
col_polygon           polygon,
col_circle            circle,
col_date              date,
col_time              time,
col_timezone          timetz,
col_timestamp         timestamp,
col_timestampzone     timestamptz);
```

10.1.2 MIGRATE WITH JDATATRANSFER TOOL

The following table schema is produced by ***JDatatransfer Tool*** automatically by default, which need adjusting (Please refer to the right schema in *10.1.3 chapter* which has been adjusted by hand).

```
create table POSTGRES_ALL_TYPES (
 COL_SMALLINT          SMALLINT       default null ,
 COL_INTEGER           INTEGER        default null ,
 COL_BIGINT            INTEGER        default null ,
 COL_SERIAL            INTEGER        not null  ,
 COL_BIGSERIAL         INTEGER        not null  ,
 COL_NUMERIC           DECIMAL(13,3)  default null ,
 COL_REAL              REAL           default null ,
 COL_DOUBLEPRECISION   REAL           default null ,
 COL_BIT               NVARCHAR(254)  default null ,
 COL_BITVARYING        NVARCHAR(254)  default null ,
 COL_BOOLEAN           CHAR(1)        default null ,
 COL_CHARACTER         NCHAR(30)      default null ,
```

```
COL_CHARACTERVARYING NVARCHAR(30)   default null ,
COL_TEXT             NCLOB          default null ,
COL_BYTEA            BINARY(254)    default null ,
COL_CIDR             NVARCHAR(254)  default null ,
COL_INET             NVARCHAR(254)  default null ,
COL_INTERVAL         NVARCHAR(254)  default null ,
COL_MACADDR          NVARCHAR(254)  default null ,
COL_MONEY            REAL           default null ,
COL_POINT            NVARCHAR(254)  default null ,
COL_LINE             NVARCHAR(254)  default null ,
COL_LSEG             NVARCHAR(254)  default null ,
COL_BOX              NVARCHAR(254)  default null ,
COL_PATH             NVARCHAR(254)  default null ,
COL_POLYGON          NVARCHAR(254)  default null ,
COL_CIRCLE           NVARCHAR(254)  default null ,
COL_DATE             DATE           default null ,
COL_TIME             TIME           default null ,
COL_TIMEZONE         NVARCHAR(254)  default null ,
COL_TIMESTAMP        TIMESTAMP      default null ,
COL_TIMESTAMPZONE    TIMESTAMP      default null )
in DEFTABLESPACE  lock mode row  fillfactor 100 ;
```

10.1.3 MODIFY TABLE SCHEMA MANUALLY

DBMaster doesn't support the BIGINT and BIGSERIAL types, so we can only replace them with INTEGER Type (will be supported in Version 5.2).

In addition, DBMaster also doesn't support box, cidr, inet, interval, macaddr and ***Geometric* Type** (*point, line, lseg, box, path, polygon, circle*); we don't plan to getting to the bottom of these types, so totally replace them with ***varchar(254)***.

```
create table postgres_all_types (
 col_smallint         smallint,
 col_integer          integer,
 col_bigint           integer,
 col_serial           integer not null,
 col_bigserial        integer not null,
 col_numeric          decimal(13, 3),
 col_real             real,
 col_doubleprecision  real,
 col_bit              char(20),
 col_bitvarying       varchar(20),
 col_boolean          char(1),
 col_character        char(30),
 col_charactervarying varchar(30),
 col_text             nclob,
 col_bytea            binary(254),
 col_cidr             varchar(254),
 col_inet             varchar(254),
 col_interval         varchar(254),
 col_macaddr          varchar(254),
 col_money            real,
 col_point            varchar(254),
```

```
 col_line            varchar(254),
 col_lseg            varchar(254),
 col_box             varchar(254),
 col_path            varchar(254),
 col_polygon         varchar(254),
 col_circle          varchar(254),
 col_date            date,
 col_time            varchar(16),
 col_timezone        varchar(23),
 col_timestamp       varchar(27),
 col_timestampzone   varchar(34));
```

Note: Time and Timestamp have the corresponding types in DBMaster, but with different precision. So we replace them with varchar(x) to avoid losing the accuracy.

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 imported from ODBC via ***JDataTransfer Tool***; Or only being exported with *TEXT-Format* file from PostgreSQL via the *Third-Part* tools (***PostgreSQL Data Wizard***), then 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 (***Import from ODBC*** in ***JDataTransfer Tool*** can work).

10.2.1 ORDINARY CHARACTER AND NUMERIC DATA TYPE

Step 1: Create table *ordinary_types* in PostgreSQL.

```
create table ordinary_types(
 col_smallint          int2,
 col_integer           int,
 col_bigint            int8,
 col_serial            serial4,
 col_bigserial         serial8,
 col_numeric           decimal(13,3),
 col_real              float4,
 col_doubleprecision   float8,
 col_character         char(30),
 col_charactervarying  varchar(30));
```

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

For example:

```
insert into ordinary_types
values(100, 666666, 22222222, 1234, 12345678, 3456.123, 3456.4, 6789.8,'col_character1','col_charactervarying1');

insert into ordinary_types
values(200, 777777, 33333333, 3456, 34567890, 4567.234, 4556.4, 5678.5,'col_character2','col_charactervarying2');

insert into ordinary_types
values(300, 888888, 44444444, 6789, 56789012, 5678.456, 5667.4, 2344.3,'col_character3','col_charactervarying3');
```

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_integer          integer,
 col_bigint           integer,
 col_serial           integer not null,
 col_bigserial        integer not null,
 col_numeric          decimal(13, 3),
 col_real             real,
 col_doubleprecision  real,
 col_character        char(30),
 col_charactervarying varchar(30));
```

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

Please refer to *Section 4.2.1 PostgreSQL Data Wizard* and export the Data with TEXT formats from PostgreSQL.

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

Please import into DBMaster via **JDATA Transfer Tool** which described in *chapter 4.2.1.2* 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 PostgreSQL.

```
create table special_types(
 col_bit              bit(20),
 col_bitvarying       varbit(20),
 col_boolean          bool,
 col_text             text,
 col_bytea            bytea,
 col_cidr             cidr,
```

```
col_inet            inet,
col_interval        interval,
col_macaddr         macaddr,
col_money           money,
col_point           point,
col_line            line,
col_lseg            lseg,
col_box             box,
col_path            path,
col_polygon         polygon,
col_circle          circle,
col_date            date,
col_time            time,
col_timezone        timetz,
col_timestamp       timestamp,
col_timestampzone   timestamptz);
```

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

For example:

```
insert into special_types
values(B'10101010101111100001',B'101010111110001101',false,'col_text1','asdw123414','10.1/16','190.168.70.253','1
235465495','45:00:87:00:40:99','98.58','(12,45)',null,'(12,45),(25,78)','(12,45),(25,78)','(1,1),(1,2),(2,2),(4,5)
','(1,1),(1,2),(2,2),(2,1)','((5,7),9)','1900-05-15','15:26:15.1234567','15:26:15.1234567 +12:15','2007-05-08
12:35:29.1234567','2007-05-08 12:35:29.1234567+14:58');

insert into special_types
values(B'10101010101111100001',B'101010111110001101',true,'col_text2','erter23543','10.1/16','191.168.70.254','12
35465496','45:00:87:00:40:99','98.58','(12,45)',null,'(12,45),(25,78)','(12,45),(25,78)','(1,1),(1,2),(2,2),(4,5)'
,'(1,1),(1,2),(2,2),(2,1)','((5,7),9)','1981-2-3','15:26:15.1234567','15:26:15.1223567+11:15','2007-05-08
12:35:29.3453565','2007-05-08 12:35:29.1234567+13:54');

insert into special_types
values(B'10101010101111100001',B'101010111110001101','1','col_text3','fghfg34534','10.1/16','192.168.70.255','123
5465497','45:00:87:00:40:99','98.58','(12,45)',null,'(12,45),(25,78)','(12,45),(25,78)','(1,1),(1,2),(2,2),(4,5)',
'(1,1),(1,2),(2,2),(2,1)','((5,7),9)','2010-05-25','15:26:15.1234567','15:26:15.1233467 +10:13','2007-05-08
12:35:29.345354','2007-05-08 12:35:29.1234567+12:50');
```

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_bit            char(20),
 col_bitvarying     varchar(20),
 col_boolean        char(1),
 col_text           nclob,
 col_bytea          binary(254),
 col_cidr           varchar(254),
 col_inet           varchar(254),
 col_interval       varchar(254),
 col_macaddr        varchar(254),
 col_money          real,
 col_point          varchar(254),
 col_line           varchar(254),
 col_lseg           varchar(254),
 col_box            varchar(254),
 col_path           varchar(254),
```

```
col_polygon          varchar(254),
col_circle           varchar(254),
col_date             date,
col_time             varchar(16),
col_timezone         varchar(23),
col_timestamp        varchar(27),
col_timestampzone    varchar(34));
```

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)

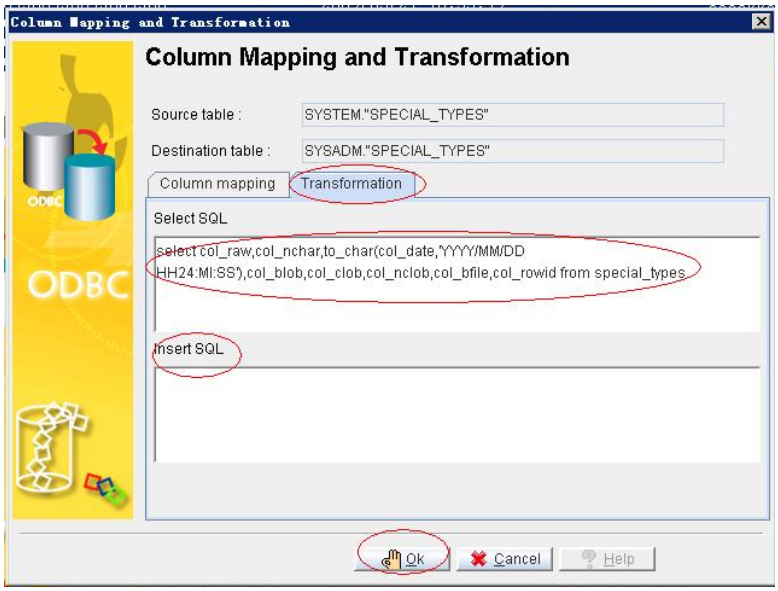
Step A: Click on the Tab Transformation and input the ***Select SQL*** for getting the result from PostgreSQL.

```
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 types.

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 PostgreSQL.

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 PostgreSQL (Npgsql .NET *Data Provider for PostgreSQL*).

10.3.1 JAVA LANGUAGE

- PostgreSQL

```
try{
    ......
    Class.forName("org.postgresql.Driver").newInstance();
    Connection conn =
DriverManager.getConnection("jdbc:postgresql://:localhost:5432:postgres", "postgres",
"postgres");
    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://127.0.0.1:2453/DBSAMPLE5","SYSADM","");
    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

- PostgreSQL(Npgsql .NET data Provider FOR Postgresql)

```
String strConn = "Server=127.0.0.1;Port=5432;User
Id=postgres;Password=postgres;Database=postgres;";
NpgsqlConnection conn = new NpgsqlConnection(strConn);
NpgsqlCommand cmd = new NpgsqlCommand();
conn.Open();
cmd.Connection = conn;
cmd.CommandText = "insert into ordinary_types(col_integer, col_char, col_float)
values(:p1,:p2,:p3)";
cmd.Parameters .Add (new NpgsqlParameter (":p1",NpgsqlTypes.NpgsqlDbType.Integer));
cmd.Parameters .Add (new NpgsqlParameter (":p2",NpgsqlTypes.NpgsqlDbType.Char ,30));
```

```
cmd.Parameters .Add (new NpgsqlParameter (":p3",NpgsqlTypes.NpgsqlDbType .Numeric));
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";
NpgsqlDataReader reader = cmd.ExecuteReader();
while(reader.Read()){
for(int i=0;i<reader.FieldCount;i++){
Console.WriteLine(reader [i]);
            }
        }
conn.Close();
```

- PostgreSQL(PostgreSQL ODBC Driver (psqlODBC))

```
string connStr = "Driver={PostgreSQL
UNICODE};Server=127.0.0.1;Port=5432;Database=postgres;Uid=postgres;Pwd=postgres;";
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 = "Hello CONN!";
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();
```

- DBMaster (*ADO.NET ODBC Provider*)

```
String connStr = "Driver={DBMaster 5.1 Driver}; Database = DBSAMPLE5;UID = SYSADM;PWD =";
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 ***.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 = DBSAMPLE5;User Id=SYSADM;Password=;";
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.

- PostgreSQL

```
<?php
try{
   $dbh = new PDO("pgsql:dbname=postgres;", "postgres", "postgres");
   /*** 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= dbsample5", "sysadm", "");
    /*** echo a message saying we have connected ***/
    echo 'Connected to database';
 }catch(PDOException $e){
    echo $e->getMessage();
 }
?>
```

Note: If users don't use the PDO in PostgreSQL as following:

```
<?php
$dbconn = pg_connect("host=localhost dbname=postgres user=postgres password=postgres");
if(!$dbconn)
{die('Could not connect: ' . pg_last_error());}
$query = 'SELECT * FROM test';
$result = pg_query($query) or die('Query failed: ' . pg_last_error());
echo "<table>\n";
while ($line = pg_fetch_array($result, null, PGSQL_ASSOC)) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
}
echo "</table>\n";
pg_free_result($result);
pg_close($dbconn);?>
```

Please use PHP ODBC API for DBMaster.

```
<?php
$conn=odbc_connect("dbsample5","SYSADM","");
if(!$conn)
{exit("Connection Failed:" . $conn);}
$rs_count=odbc_exec($conn,"select count(*) from sysuser");
if(!$rs_count)
{exit("Error in SQL");}
echo "<table><tr>";
echo "<th>count</th></tr>";
$col=odbc_result($rs_count,1);
echo "<tr><td>$col</td></tr>";
odbc_close($conn);
echo"</table>";
?>
```