Postgres Plus® Migration Guide

Postgres Plus Advanced Server 9.3

November 25, 2015

Postgres Plus Migration Guide, Version 3.1
by EnterpriseDB Corporation

EnterpriseDB Corporation, 34 Crosby Drive, Suite 100, Bedford, MA 01730, USA
T +1 781 357 3390 **F** +1 978 467 1307 **E** info@enterprisedb.com **www**.enterprisedb.com

Table of Contents

1 Introduction 5
1.1 Typographical Conventions Used in this Guide 6
2 Migration Methodology 7
2.1 The Migration Process 7
2.2 Connecting an Application to Postgres 9
3 Functionality Overview 11
4 Installing Migration Toolkit 13
4.1 Installing Migration Toolkit with Advanced Server 13
4.2 Using Stack Builder to Install Migration Toolkit 14
4.3 Installing Source-Specific Drivers 27
5 Building the toolkit.properties File 28
5.1 Defining an Advanced Server URL 30
5.2 Defining a PostgreSQL URL 32
5.3 Defining an Oracle URL 34
5.4 Defining a MySQL URL 36
5.5 Defining a Sybase URL 37
5.6 Defining a SQL Server URL 38
6 Executing Migration Toolkit 40
6.1 Migrating a Schema from Oracle 41
6.2 Migrating from a Non-Oracle Source Database 42
7 Migration Toolkit Command Options 44
7.1 Offline Migration 45
7.1.1 Executing Offline Migration Scripts 46
7.2 Import Options 47
7.3 Schema Creation Options 48
7.4 Schema Object Selection Options 48
7.5 Migration Options 51
7.6 Oracle Specific Options 54
7.7 Miscellaneous Options 56
8 Migration Issues 58
8.1 Migration Toolkit Connection Errors 58
8.1.1 Invalid username/password 58

8.1.2 Connection rejected: FATAL: password 59
8.1.3 Exception: ORA-28000: the account is locked 59
8.1.4 Exception: oracle.jdbs.driver.OracleDriver 59
8.1.5 I/O exception: The Network Adapter could not establish the connection 59
8.1.6 Exception: The connection attempt failed 60
8.2 Migration Toolkit Migration Errors 61
8.2.1 ERROR: Extra Data after last expected column 61
8.2.2 Error Loading Data into Table: *TABLE_NAME*: null 61
8.2.3 Error Creating Constraint `CONS_NAME_FK` 61
8.2.4 Error Loading Data into Table 62
8.2.5 ERROR: value too long for type 62
8.2.6 ERROR: Exception on Toolkit thread:OutOfMemoryError 63
8.3 Unsupported Oracle Features 64
8.4 Frequently Asked Questions 65

1 Introduction

Migration Toolkit is a powerful command-line tool that offers granular control of the migration process. Migration Toolkit facilitates migration of database objects and data to an Advanced Server or PostgreSQL database from:

- Oracle
- MySQL
- SQL Server

Migration Toolkit also allows you to migrate from Sybase to Advanced Server, or between Advanced Server and PostgreSQL.

You can install Migration Toolkit with the Postgres Plus Advanced Server installer, or via Stack Builder. Stack Builder is distributed with both Advanced Server, and the PostgreSQL one-click installer available from the EnterpriseDB web site at:

http://www.enterprisedb.com/downloads/postgres-postgresql-downloads

This guide provides a high-level description of the steps involved in the migration process, as well as installation and usage instructions for Migration Toolkit. It also includes solutions to common migration problems, and details unsupported features and their potential workarounds.

Please Note:

This guide uses the term Postgres to refer to *either* an installation of Postgres Plus Advanced Server or PostgreSQL.

This guide uses the term Stack Builder to refer to *either* StackBuilder Plus (distributed with Postgres Plus Advanced Server) or Stack Builder (distributed with the PostgreSQL one-click installer from EnterpriseDB).

EnterpriseDB does not support the use of Migration Toolkit with Oracle Real Application Clusters (RAC) and Oracle Exadata; the aforementioned Oracle products have not been evaluated nor certified with this EnterpriseDB product.

1.1 Typographical Conventions Used in this Guide

Certain typographical conventions are used in this manual to clarify the meaning and usage of various commands, statements, programs, examples, etc. This section provides a summary of these conventions.

In the following descriptions a *term* refers to any word or group of words that are language keywords, user-supplied values, literals, etc. A term's exact meaning depends upon the context in which it is used.

- *Italic font* introduces a new term, typically, in the sentence that defines it for the first time.
- `Fixed-width (mono-spaced) font` is used for terms that must be given literally such as SQL commands, specific table and column names used in the examples, programming language keywords, etc. For example, `SELECT * FROM emp;`
- *`Italic fixed-width font`* is used for terms for which the user must substitute values in actual usage. For example, `DELETE FROM` *`table_name`*;
- A vertical pipe | denotes a choice between the terms on either side of the pipe. A vertical pipe is used to separate two or more alternative terms within square brackets (optional choices) or braces (one mandatory choice).
- Square brackets [ ] denote that one or none of the enclosed term(s) may be substituted. For example, `[ a | b ]`, means choose one of "`a`" or "`b`" or neither of the two.
- Braces {} denote that exactly one of the enclosed alternatives must be specified. For example, `{ a | b }`, means exactly one of "`a`" or "`b`" must be specified.
- Ellipses ... denote that the proceeding term may be repeated. For example, `[ a | b ] ...` means that you may have the sequence, "`b a a b a`".

2 Migration Methodology

There are many reasons to consider migrating from one database to another. Migration can allow you to take advantage of new or better technology. If your current database does not offer the right set of capabilities to allow you to scale the system, moving to a database that offers the functionality you need is the best move for your company.

Migration can also be very cost effective. Migrating systems with significant maintenance costs can save money spent on system upkeep. By consolidating the number of databases in use, you can also reduce in-house administrative costs. By using fewer database platforms (or possibly taking advantage of database compatibility), you can do more with your IT budget.

Using more than one database platform can offer you a graceful migration path should a vendor raise their pricing or change their company directive. EnterpriseDB has helped companies migrate their existing database systems to Postgres for years.

We recommend following the methodology detailed in Section 2.1, *The Migration Process*.

2.1 The Migration Process

The migration path to Postgres includes the following main steps:

1. Start the migration process by determining which database objects and data will be included in the migration. Form a migration team that includes someone with solid knowledge of the architecture and implementation of the source system.

2. Identify potential migration problems. If it is an Oracle-to-Advanced Server migration, consult the Database Compatibility for Oracle Developer's Guide for complete details about the compatibility features supported in Advanced Server. Consider using EnterpriseDB's migration assessment service to assist in this review.

3. Prepare the migration environment. Obtain and install the necessary software, and establish connectivity between the servers.

4. If the migration involves a large body of data, consider migrating the schema definition before moving the data. Verify the results of the DDL migration and resolve any problems reported in the migration summary. Section 8 of this document includes information about resolving migration problems.

5. Migrate the data. For small data sets, use Migration Toolkit. If it is an Oracle migration (into Advanced Server), and the data set is large or if you notice slow data transfer, take advantage of one of the other data movement methods available:

    - Use the Advanced Server Oracle-compatible database link feature.

    - If your data has BLOB or CLOB data, use the `dblink_ora` style database links instead of the Oracle style database links.

    Both of these methods use the Oracle Call Interface (OCI) to connect to Oracle. After connecting, use an SQL statement to select the data from the 'linked' Oracle database and insert the data into the Postgres Plus Advanced Server database.

6. Confirm the results of the data migration and resolve any problems reported in the migration summary.

7. Convert applications to work with the newly migrated Postgres database. Applications that use open standard connectivity such as JDBC or ODBC normally only require changes to the database connection strings and selection of the EnterpriseDB driver. See Section 2.2, *Connecting an Application to Postgres* for more information.

8. Test the system performance, and tune the new server. If you are migrating into an Advanced Server database, take advantage of Advanced Server's performance tuning utilities:

    - Use Dynatune to dynamically adjust database configuration resources.

    - Use Optimizer Hints to direct the query path.

    - Use the `ANALYZE` command to retrieve database statistics.

The Postgres Plus Advanced Server Performance Features Guide and Database Compatibility for Oracle Developer's Guide (both available through the EnterpriseDB website) offer information about the performance tuning tools available with Advanced Server.

2.2 Connecting an Application to Postgres

To convert a client application to use a Postgres database, you must modify the connection properties to specify the new target database. In the case of a Java application, change the JDBC driver name (`Class.forName`) and JDBC URL.

A Java application running on Oracle might have the following connection properties:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con =
   DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
                               "user",
                               "password")
```

Modify the connection string to connect to a Postgres server:

```
Class.forName("com.edb.Driver")
Connection con =
   DriverManager.getConnection("jdbc:edb://localhost:5444/edb",
                               "user",
                               "password");
```

Converting an ODBC application to connect to an instance of Postgres is a two-step process. To connect an ODBC application:

1. Use an ODBC data source administrator to create a data source that defines the connection properties for the new target database.

Most Linux and Windows systems include graphical tools that allow you to create and edit ODBC data sources. After installing ODBC, check the `Administrative Tools` menu for a link to the `ODBC Data Source Administrator`. Click the `Add` button to start the `Create New Data Source` wizard; complete the dialogs to define the new target data source.

2. Change the application to use the new data source.

The application will contain a call to `SQLConnect` (or possibly `SQLDriverConnect`); edit the invocation to change the data source name. In the following example, the data source is named `"OracleDSN"`:

```
result = SQLConnect(conHandle,           // Connection handle (returned)
            "OracleDSN", SQL_NTS,        // Data source name
            username, SQL_NTS,           // User name
            password, SQL_NTS);          // Password
```

To connect to an instance of Postgres defined in a data source named `"PostgresDSN"`, change the data source name:

```
result = SQLConnect(conHandle,          // Connection handle (returned)
            "PostgresDSN", SQL_NTS,     // Data source name
```

```
            username, SQL_NTS,          // User name
            password, SQL_NTS);         // Password
```

After establishing a connection between the application and the server, test the application to find any compatibility problems between the application and the migrated schema. In most cases, a simple change will resolve any incompatibility that the application encounters. In cases where a feature is not supported, use a workaround or third party tool to provide the functionality required by the application. See Section 8, *Migration Issues*, for information about some common problems and their workarounds.

3 Functionality Overview

Migration Toolkit is a powerful command-line tool that offers granular control of the migration process. Migration Toolkit includes a number of options, allowing you granular control of the migration process:

- Use the `-safeMode` option to commit each row as it is migrated.
- Use the `-fastCopy` option to bypass WAL logging to optimize migration.
- Use the `-batchSize` option to control the batch size of bulk inserts.
- Use the `-cpBatchSize` option to specify the batch size used with the `COPY` command.
- Use the `-filterProp` option to migrate only those rows that meet a user-defined condition.
- Use the `-customColTypeMapping` option to change the data type of selected columns.
- Use the `-dropSchema` option to drop the existing schema and create a new schema prior to migration.
- On Advanced Server, use the `-allDBLinks` option to migrate all Oracle database links.
- On Advanced Server, use the `-copyViaDBLinkOra` option to enable the `dblink_ora` module.

Using Migration Toolkit is a two-step process:

1. Edit the `toolkit.properties` file to specify the source and target database.
2. Invoke Migration Toolkit at the command line, specifying migration options.

Migration Toolkit facilitates migration of database objects and data to an Advanced Server or PostgreSQL database from:

- Oracle
- MySQL
- SQL Server

Migration Toolkit also allows you to migrate from Sybase to Advanced Server, or between Advanced Server and PostgreSQL.

For detailed information about defining `toolkit.properties` entries, please refer to Section 5, *Building the toolkit.properties File*.

Object Migration Support

Migration Toolkit migrates object definitions (DDL), table data, or both. The following table contains a platform-specific list of the types of database objects that Migration Toolkit can migrate:

Object	Oracle	Sybase	SQL Server	MySQL
Schemas	X	X	X	X
Tables	X	X	X	X
List-Partitioned Tables	X			
Range-Partitioned Table	X			
Constraints	X	X	X	X
Indexes	X	X	X	X
Triggers	X			
Table Data	X	X	X	X
Views	X		X	
Materialized Views	X			
Packages	X			
Procedures	X			
Functions	X			
Sequences	X			
Users/Roles	X			
Object Types	X			
Object Type Methods	X			
Database Links	X			

For detailed information about the commands that offer granular control of the objects imported, please see Section 7.4, *Schema Object Selection Options*.

Online Migration vs. Offline Migration

Migration Toolkit can migrate immediately and directly into a Postgres database (*online migration*), or you can also choose to generate scripts to use at a later time to recreate object definitions in a Postgres database (*offline migration*).

By default, Migration Toolkit creates objects directly into a Postgres database; in contrast, include the `-offlineMigration` option to generate SQL scripts you can use at a later time to reproduce the migrated objects or data in a new database. You can alter migrated objects by customizing the migration scripts generated by Migration Toolkit before you execute them. With the `-offlineMigration` option, you can schedule the actual migration at a time that best suits your system load.

For more information about the `-offlineMigration` option, see Section 7.1, *Offline Migration*.

4 Installing Migration Toolkit

You can use the Postgres Plus Advanced Server installer or Stack Builder to install Migration Toolkit. Stack Builder is distributed with both Advanced Server and the PostgreSQL one-click installer, available from EnterpriseDB. For more information about performing an installation with Stack Builder, see Section 4.2.

Before installing Migration Toolkit, you must first install Java (version 1.6.0 or later). Free downloads of Java installers and installation instructions are available at:

http://www.java.com/en/download/index.jsp

4.1 Installing Migration Toolkit with Advanced Server

To ensure the installation of Migration Toolkit by the Postgres Plus Advanced Server installation wizard, confirm that the `Migration Toolkit` option is checked on the `Select Components` dialog (as shown below, in Figure 4.1).

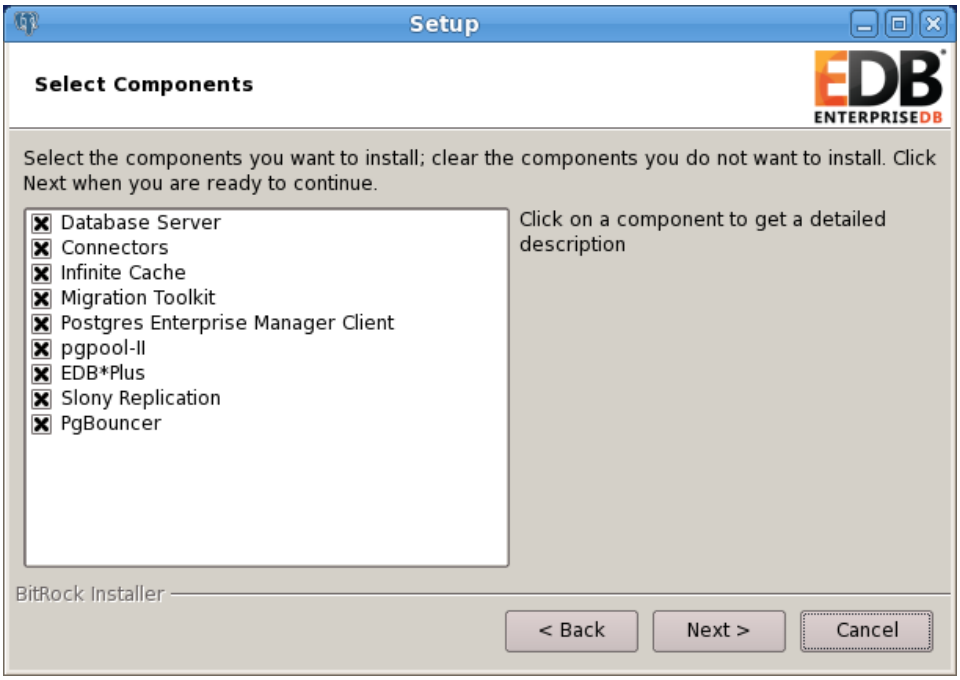

Figure 4.1 - The Select Components dialog from the Advanced Server Installation Wizard.

After installing Migration Toolkit with Advanced Server, you must install the appropriate source-specific drivers before performing a migration; please see Section 4.3 for more information.

4.2 Using Stack Builder to Install Migration Toolkit

Please Note: This guide uses the term Stack Builder to refer to *either* StackBuilder Plus (distributed with Postgres Plus Advanced Server) or Stack Builder (distributed with the PostgreSQL one-click installer from EnterpriseDB).

You can use the Stack Builder installation wizard to install Migration Toolkit into either PostgreSQL or Advanced Server.

Launching StackBuilder Plus from Advanced Server

The Advanced Server installation wizard will offer to open StackBuilder Plus when the installation completes. To launch StackBuilder Plus from an existing Advanced Server installation, navigate through the `Start` (or `Applications`) menu to the `Postgres Plus Advanced Server` menu, and select the `StackBuilder Plus` menu option.

Launching Stack Builder from PostgreSQL

If you are installing PostgreSQL using the EnterpriseDB one-click installer, the installer will offer to open Stack Builder when the installation completes. Please note that you must have a Java JVM (version 1.6.0 or later) in place before Stack Builder can perform a Migration Toolkit installation.

Free downloads of Java installers and installation instructions are available at:

http://www.java.com/en/download/index.jsp

The Java executable must be in your search path (`%PATH%` on Windows, `$PATH` on Linux/Unix). Use the following commands to set the search path (substituting the name of the directory that holds the Java executable for *`javadir`*):

On Windows, use the command:

```
SET PATH=javadir;%PATH%
```

On Linux, use the command:

```
PATH=javadir:$PATH
```

To open Stack Builder, navigate through the `Start` (or `Applications`) menu to the `PostgreSQL` installation menu, and select the `Application Stack Builder` menu option. Stack Builder opens as shown in Figure 4.2.

Figure 4.2 - The Stack Builder welcome window.

Use the drop-down listbox to select the target server installation from the list of available servers. If your network requires you to use a proxy server to access the Internet, use the `Proxy servers` button to open the `Proxy servers` dialog and specify a proxy server; if you do not need to add a proxy server, click `Next` to open the application selection window.

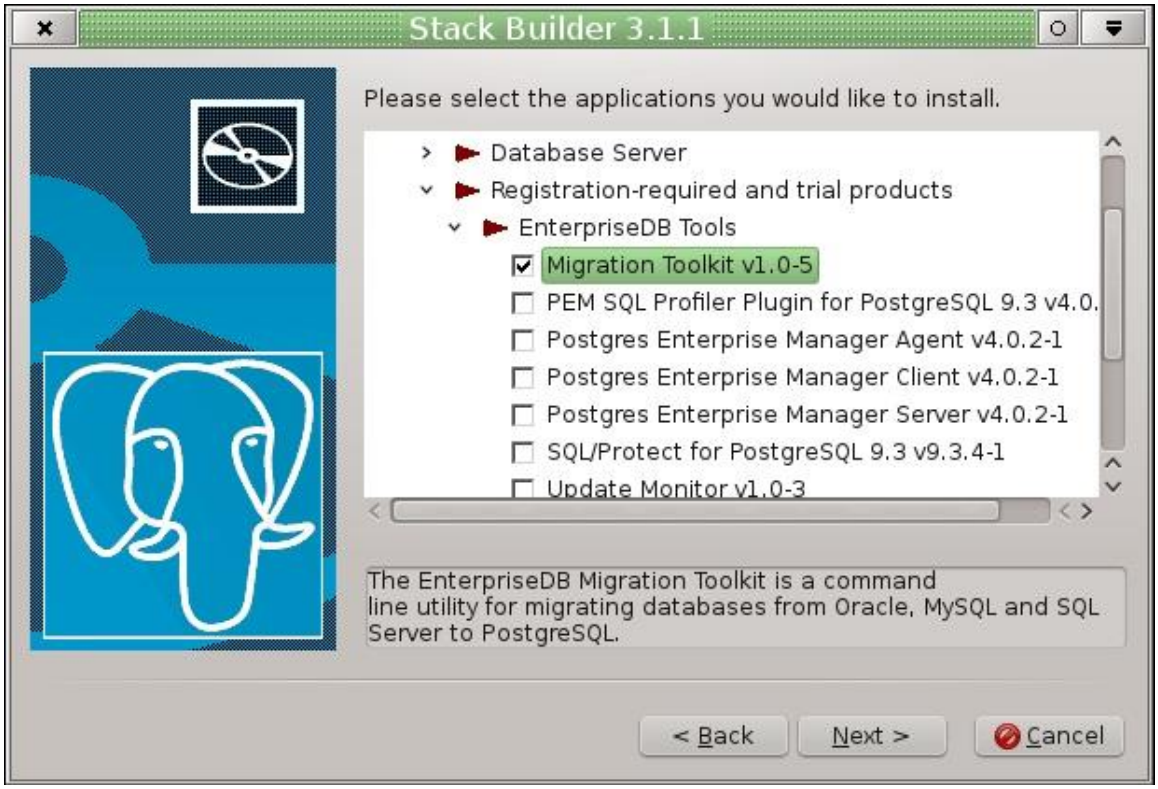

Figure 4.3 - The Stack Builder application selection window.

If you are using Stack Builder to add Migration Toolkit to your PostgreSQL installation, expand the `EnterpriseDB Tools` node of the tree control (located under the `Registration-required and trial products` node), and check the box next to `Migration Toolkit v1.0-4` (as shown in Figure 4.3). Click `Next` to continue.

If you are using StackBuilder Plus to add Migration Toolkit to your Advanced Server installation, expand the `Add-ons, tools and utilities` node of the tree control, and check the box next to `EnterpriseDB Migration Toolkit`. Click `Next` to continue.

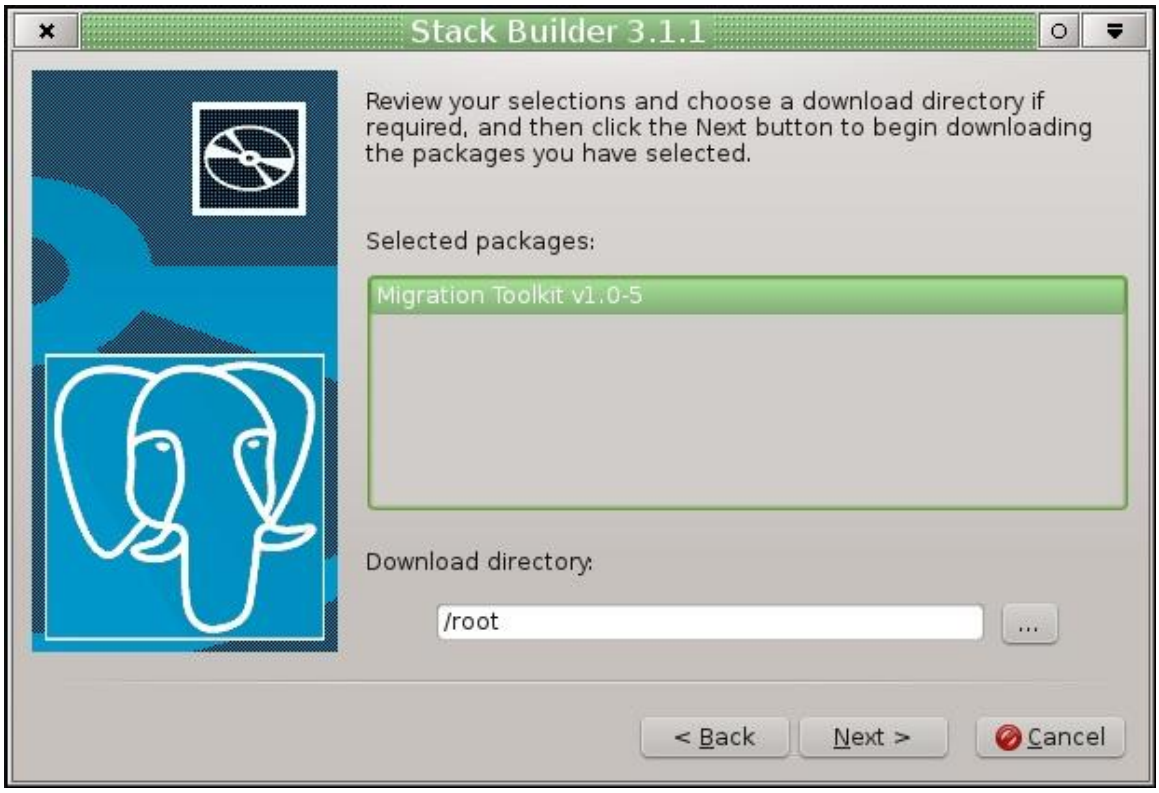

Figure 4.4 - The Stack Builder selection confirmation window.

Confirm that Migration Toolkit is included in the `Selected Packages` list and that the `Download directory` field contains an acceptable download location (as shown in Figure 4.4). Click `Next` to start the Migration Toolkit download.

Figure 4.5 - Stack Builder confirms the file download.

When the download completes, Stack Builder confirms that the installation files have been successfully downloaded (Figure 4.5). Choose `Next` to open the Migration Toolkit installation wizard .

Figure 4.6 - The Migration Toolkit installation wizard.

Select an installation language from the drop-down listbox, and click `OK`. The Migration Toolkit installation wizard opens (as shown in Figure 4.6). Click `Next` to continue.

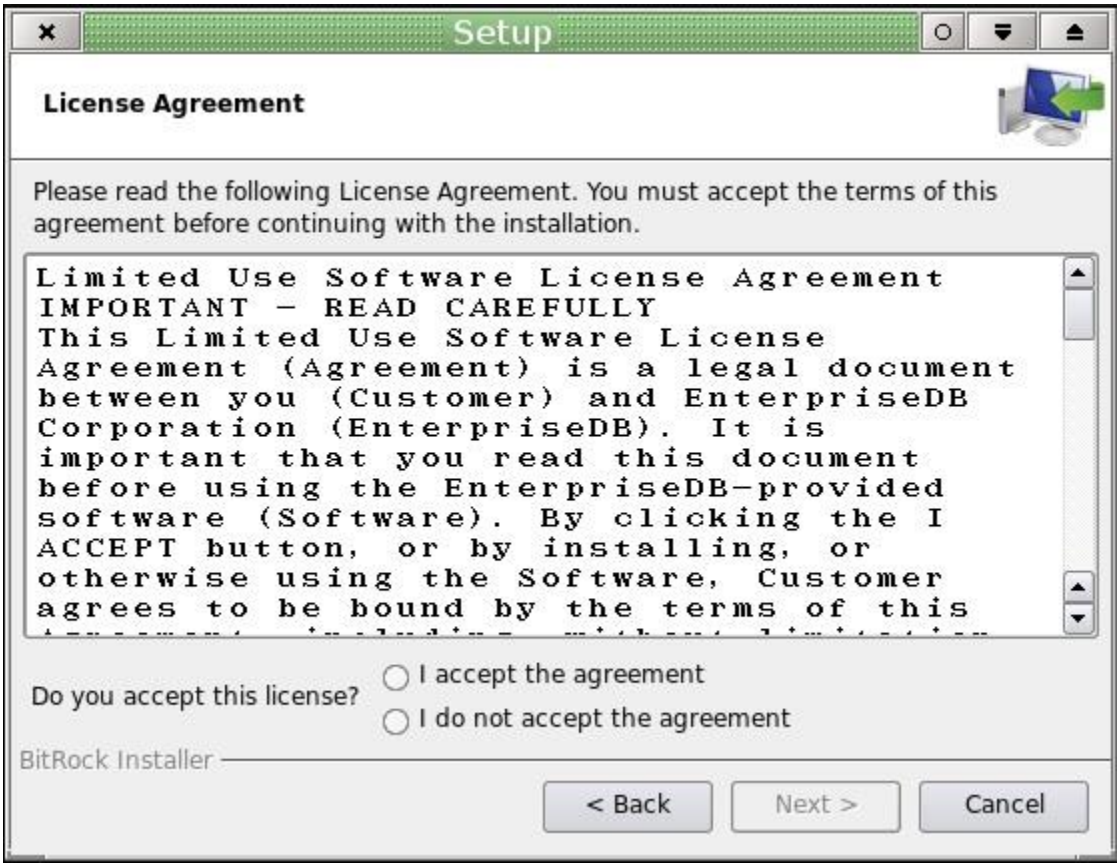

Figure 4.7 - The EnterpriseDB license agreement.

Carefully review the license agreement (as shown in Figure 4.7) before highlighting the appropriate radio button; click `Next` to continue to the user registration window (shown in Figure 4.8).

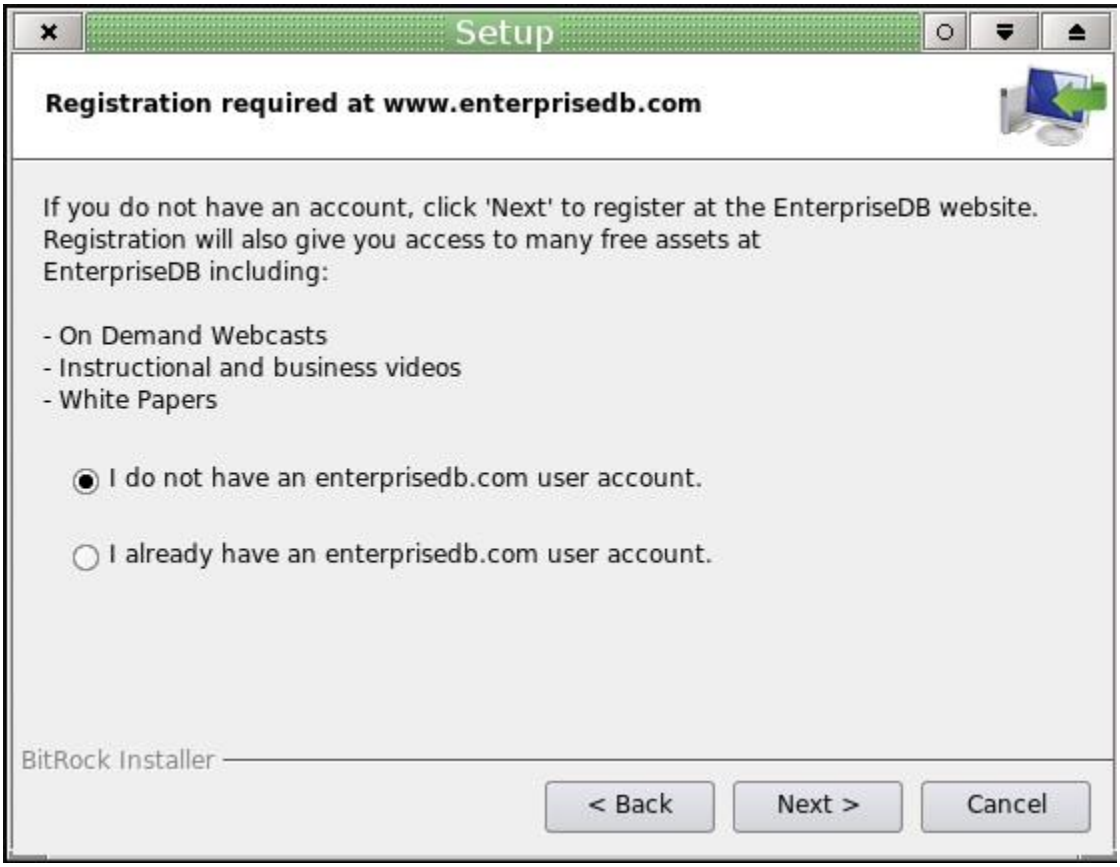

Figure 4.8 - Registration is required before installation.

To install Migration Toolkit, you must provide the email address and password associated with your EnterpriseDB user account. Registration is free, and provides access to many EnterpriseDB resources, including the Migration Toolkit installer.

- If you are currently an EnterpriseDB registered user, click the radio button next to `I already have an enterprisedb.com user account` to proceed to the EnterpriseDB User Account information window (shown in Figure 4.10).

- If you do not have an EnterpriseDB user account, click the radio button next to `I do not have an enterprisedb.com user account` to open a browser (shown in Figure 4.9), and provide registration information.

When you've selected the appropriate radio button, click `Next` to continue.

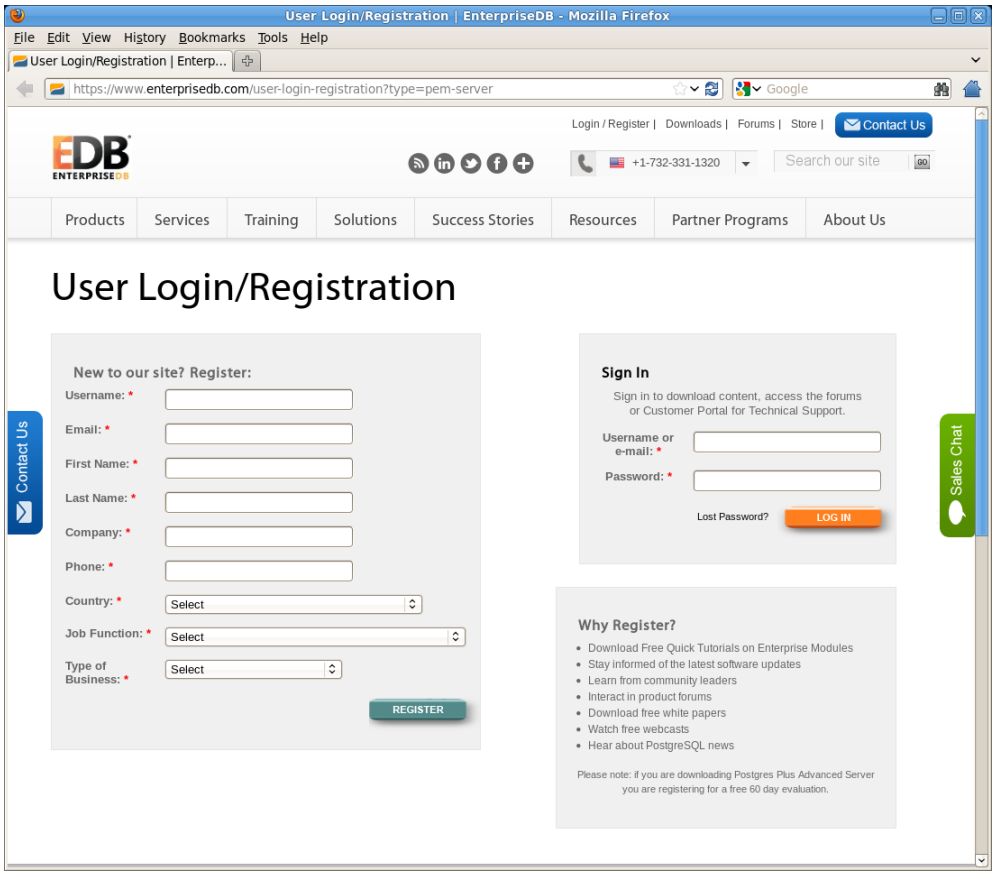

Figure 4.9 - The User Login/Registration page.

If you are not a registered EnterpriseDB user, complete the `User Login/Registration` portion of the window, and click `Next` to continue.

Figure 4.10 - The EnterpriseDB User Account Information window.

Enter the email address and password associated with your EnterpriseDB user account, and click `Next` to continue.

Figure 4.11 - Specify a Migration Toolkit installation directory.

By default, Migration Toolkit will be installed under the PostgresPlus directory (see Figure 4.11). Accept the default installation directory, or change the directory, and click `Next` to continue.

Figure 4.12 - Completing the Migration Toolkit installation.

The installation wizard confirms that the `Setup` program is ready to install Migration Toolkit (as shown in Figure 4.12); click `Next` to start the installation.

Figure 4.13 - Stack Builder confirms the installation is complete.

A dialog confirms that the Migration Toolkit installation is complete (see Figure 4.13); click `Finish` to exit the installer.

Figure 4.14 - Stack Builder confirms the installation is complete.

When Stack Builder finalizes installation of the last selected component, it displays the `Installation Completed` window (shown in Figure 4.14). Click `Finish` to close Stack Builder.

After installing Migration Toolkit with Stack Builder, you must install the appropriate source-specific drivers before performing a migration; please see Section 4.3 for more information.

4.3 Installing Source-Specific Drivers

Before invoking Migration Toolkit, you must download and install a freely available source-specific driver. To download a driver, or for a link to a vendor download site, visit the `Third Party JDBC Drivers` page at the EnterpriseDB website:

http://www.enterprisedb.com/downloads/third-party-jdbc-drivers

After downloading the source-specific driver, move the driver file into the `JAVA_HOME/jre/lib/ext` directory.

5 Building the toolkit.properties File

Migration Toolkit uses the configuration and connection information stored in the `toolkit.properties` file during the migration process to identify and connect to the source and target databases. By default:

The Advanced Server `toolkit.properties` file is located (on Linux) in:

```
/opt/PostgresPlus/9.3AS/etc
```

and on Windows in:

```
C:\Program Files\PostgresPlus\9.3AS\etc
```

The PostgreSQL `toolkit.properties` file is located (on Linux) in:

```
/opt/PostgresPlus/EnterpriseDB-MigrationToolkit
```

and on Windows in:

```
C:\Program Files\PostgresPlus\EnterpriseDB-MigrationToolkit
```

A sample `toolkit.properties` file is shown in Figure 5.1.

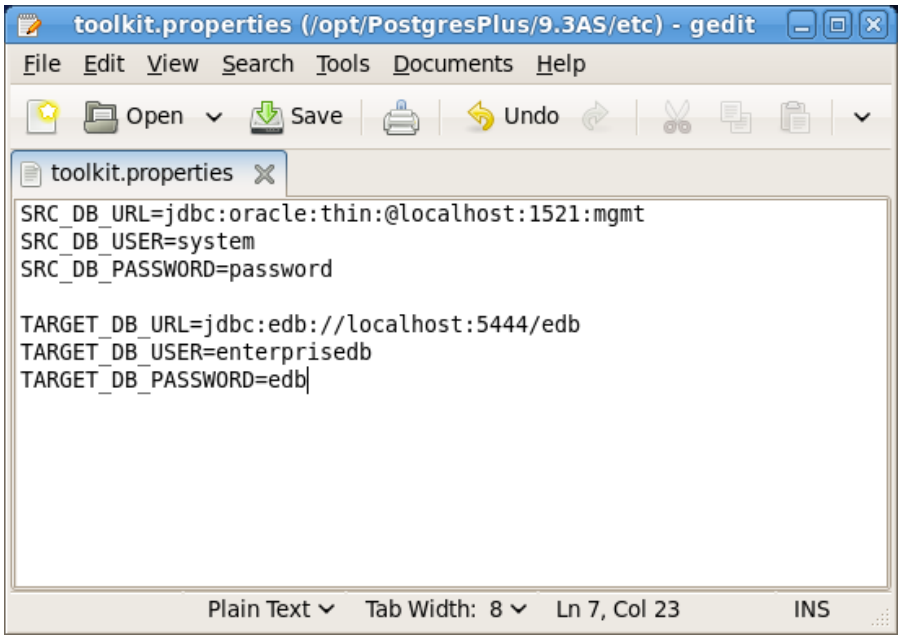

Figure 5.1 - A typical `toolkit.properties` *file.*

Before executing Migration Toolkit commands, modify the `toolkit.properties` file with the editor of your choice. Update the file to include the following information:

- `SRC_DB_URL` specifies how Migration Toolkit should connect to the source database. See the section corresponding to your source database for details about forming the URL.
- `SRC_DB_USER` specifies a user name (with sufficient privileges) in the source database.
- `SRC_DB_PASSWORD` specifies the password of the source database user.
- `TARGET_DB_URL` specifies the JDBC URL of the target database.
- `TARGET_DB_USER` specifies the name of a privileged target database user.
- `TARGET_DB_PASSWORD` specifies the password of the target database user.

5.1 Defining an Advanced Server URL

Migration Toolkit facilitates migration from the following platforms to Advanced Server:

- Oracle
- MySQL
- Sybase
- SQL Server
- PostgreSQL

For a definitive list of the objects migrated from each database type, please refer to Section 3, *Functionality Overview*.

Migration Toolkit reads connection specifications for the source and the target database from the `toolkit.properties` file. Connection information for each must include:

- The URL of the database
- The name of a privileged user
- The password associated with the specified user.

The URL conforms to JDBC standards and takes the form:

```
{TARGET_DB_URL|SOURCE_DB_URL}=jdbc:edb://host:port/database_id
```

An Advanced Server URL contains the following information:

`jdbc`

The protocol is always `jdbc`.

`edb`

If you are using Advanced Server, specify `edb` for the sub-protocol value.

`host`

The name or IP address of the host where the Postgres instance is running.

`port`

The port number that the Advanced Server database listener is monitoring. The default port number is `5444`.

`database_id`

The name of the source or target database.

`{TARGET_DB_USER|SOURCE_DB_USER}` must specify a user with privileges to `CREATE` each type of object migrated. If migrating data into a table, the specified user may also require `INSERT`, `TRUNCATE` and `REFERENCES` privileges for each target table.

`{TARGET_DB_PASSWORD|SOURCE_DB_PASSWORD}` is set to the password of the privileged Advanced Server user.

5.2 Defining a PostgreSQL URL

Migration Toolkit facilitates migration from the following platforms to PostgreSQL:

- Oracle
- MySQL
- SQL Server
- Postgres Plus Advanced Server

For a definitive list of the objects migrated from each database type, please refer to Section 3, *Functionality Overview*.

Migration Toolkit reads connection specifications for the source and the target database from the `toolkit.properties` file. Connection information for each must include:

- The URL of the database
- The name of a privileged user
- The password associated with the specified user.

A PostgreSQL URL conforms to JDBC standards and takes the form:

`{SOURCE_DB_URL|TARGET_DB_URL}=jdbc:postgresql://`*`host`*`:`*`port/database_id`*

The URL contains the following information:

`jdbc`

The protocol is always `jdbc`.

`postgresql`

If you are using PostgreSQL, specify `postgresql` for the sub-protocol value.

`host`

The name or IP address of the host where the Postgres instance is running.

`port`

The port number that the Postgres database listener is monitoring. The default port number is `5432`.

`database_id`

The name of the source or target database.

`{SOURCE_DB_USER|TARGET_DB_USER}` must specify a user with privileges to `CREATE` each type of object migrated. If migrating data into a table, the specified user may also require `INSERT`, `TRUNCATE` and `REFERENCES` privileges for each target table.

`{SOURCE_DB_PASSWORD|TARGET_DB_PASSWORD}` is set to the password of the privileged PostgreSQL user.

5.3 Defining an Oracle URL

Migration Toolkit facilitates migration from an Oracle database to a PostgreSQL or Advanced Server database. When migrating from Oracle, you must specify connection specifications for the Oracle source database in the `toolkit.properties` file. The connection information must include:

- The URL of the Oracle database
- The name of a privileged user
- The password associated with the specified user.

When migrating from an Oracle database, `SRC_DB_URL` should contain a JDBC URL, specified in one of two forms. The first form is:

```
jdbc:oracle:thin:@host_name:port:database_id
```

The second form is:

```
jdbc:oracle:thin:@//host_name:port/{database_id|service_name}
```

An Oracle URL contains the following information:

`jdbc`

The protocol is always `jdbc`.

`oracle`

The sub-protocol is always `oracle`.

`thin`

The driver type. Specify a driver type of `thin`.

`host_name`

The name or IP address of the host where the Oracle server is running.

`port`

The port number that the Oracle database listener is monitoring.

`database_id`

The database SID of the Oracle database.

service_name

The name of the Oracle service.

SRC_DB_USER should specify the name of a privileged Oracle user.

SRC_DB_PASSWORD must contain the password of the specified user.

5.4 Defining a MySQL URL

Migration Toolkit facilitates migration from a MySQL database to an Advanced Server or PostgreSQL database. When migrating from MySQL, you must specify connection specifications for the MySQL source database in the `toolkit.properties` file. The connection information must include:

- The URL of the source database
- The name of a privileged user
- The password associated with the specified user.

When migrating from MySQL, `SRC_DB_URL` takes the form of a JDBC URL. For example:

```
jdbc:mysql://host_name[:port]/database_id
```

A MySQL URL contains the following information:

`jdbc`

The protocol is always `jdbc`.

`mysql`

The sub-protocol is always `mysql`.

`//host_name`

The name or IP address of the host where the source server is running.

`[`*`port`*`]`

The port number that the MySQL database listener is monitoring.

`/database_id`

The name of the source database.

`SRC_DB_USER` should specify the name of a privileged MySQL user.

`SRC_DB_PASSWORD` must contain the password of the specified user.

5.5 Defining a Sybase URL

Migration Toolkit facilitates migration from a Sybase database to an Advanced Server database. When migrating from Sybase, you must specify connection specifications for the Sybase source database in the `toolkit.properties` file. The connection information must include:

- The URL of the source database
- The name of a privileged user
- The password associated with the specified user.

When migrating from Sybase, `SRC_DB_URL` takes the form of a JTDS URL. For example:

```
jdbc:jtds:sybase://host_name[:port]/database_id
```

A Sybase URL contains the following information:

`jdbc`

The protocol is always `jdbc`.

`jtds`

The driver name is always `jtds`.

`sybase`

The server type is always `sybase`.

`host_name`

The name or IP address of the host where the source server is running.

`port`

The port number that the Sybase database listener is monitoring.

`database_id`

The name of the source database.

`SRC_DB_USER` should specify the name of a privileged Sybase user.

`SRC_DB_PASSWORD` must contain the password of the specified user.

5.6 Defining a SQL Server URL

Migration Toolkit facilitates migration from a SQL Server database to a PostgreSQL or Advanced Server database. Migration Toolkit supports migration of the following object definitions:

- schemas
- tables
- table data
- constraints
- indexes

Migration Toolkit reads connection specifications for the source database from the `toolkit.properties` file. The connection information must include:

- The URL of the source database
- The name of a privileged user
- The password associated with the specified user.

If you are connecting to a SQL Server database, `SRC_DB_URL` takes the form of a JTDS URL. For example:

```
jdbc:jtds:sqlserver://server[:port]/database_id
```

A SQL Server URL contains the following information:

`jdbc`

The protocol is always `jdbc`.

`jtds`

The driver name is always `jtds`.

`sqlserver`

The server type is always `sqlserver`.

`server_name`

The name or IP address of the host where the source server is running.

`port`

The port number that the source database listener is monitoring.

database_id

The name of the source database.

SRC_DB_USER should specify the name of a privileged SQL Server user.

SRC_DB_PASSWORD must contain the password of the specified user.

6 Executing Migration Toolkit

After installing Migration Toolkit, and specifying connection properties for the source and target databases in the `toolkit.properties` file, Migration Toolkit is ready to perform migrations.

The Migration Toolkit executable is named `runMTK.sh` on Linux/Unix systems and `runMTK.bat` on Windows systems.

On a Linux Advanced Server installation, the executable is located in:

```
/opt/PostgresPlus/9.3AS/bin
```

and on Windows, in:

```
C:\Program Files\PostgresPlus\9.3AS\bin
```

On a Linux PostgreSQL installation, the executable is located in:

```
/opt/PostgresPlus/EnterpriseDB-MigrationToolkit
```

and on Windows, in:

```
C:\Program Files\PostgresPlus\EnterpriseDB-
MigrationToolkit
```

Importing Character Data with Embedded Binary Zeros (NULL characters)

The Migration Toolkit properly supports importation of a column where its value is `NULL`.

However, the Migration Toolkit does not support importation of `NULL` character values (embedded binary zeros 0x00) with the JDBC connection protocol. If you are importing data that includes the `NULL` character, use the `-replaceNullChar` option to replace the `NULL` character with a single, non-`NULL`, replacement character.

Once the data has been migrated, use a SQL statement to replace the character specified by `-replaceNullChar` with binary zeros.

6.1 Migrating a Schema from Oracle

Unless specified in the command line, Migration Toolkit expects the source database to be Oracle and the target database to be Advanced Server. To migrate a complete schema on Linux, navigate to the executable and invoke the following command:

```
$ ./runMTK.sh schema_name
```

To migrate a complete schema on Windows, navigate to the executable and invoke the following command:

```
> .\runMTK.bat schema_name
```

Where:

schema_name

> *schema_name* is the name of the schema within the source database (specified in the `toolkit.properties` file) that you wish to migrate. You must include at least one *schema_name*.

You can migrate multiple schemas by following the command name with a comma-delimited list of schema names.

On Linux, execute the following command:

```
$ ./runMTK.sh schema_name1,schema_name2,schema_name3
```

On Windows, execute the following command:

```
> .\runMTK.bat schema_name1,schema_name2,schema_name3
```

6.2 Migrating from a Non-Oracle Source Database

If you do not specify a source database type and a target database type, Postgres assumes the source database to be Oracle, and the target database to be Postgres Plus Advanced Server.

To invoke Migration Toolkit, open a command window and navigate to the `migrationstudio` directory, located under your installation of Advanced Server. Invoke Migration Toolkit with the following command:

```
$ ./runMTK.sh –sourcedbtype db_type –targetdbtype
target_type [options, …] schema_name;
```

Where:

`-sourcedbtype source_type`

`source_type` specifies the server type of the source database. *`source_type`* is case-insensitive. By default, *`source_type`* is `oracle`. *`source_type`* may be one of the following values:

To migrate from:	Specify:
Oracle	`oracle` (the default value)
MySQL	`mysql`
SQL Server	`sqlserver`
Sybase	`sybase`
PostgreSQL	`postgres` or `postgresql`
Advanced Server	`enterprisedb`

`-targetdbtype target_type`

`target_type` specifies the server type of the target database. *`target_type`* is case-insensitive. By default, *`target_type`* is `enterprisedb`. *`target_type`* may be one of the following values:

To migrate to:	Specify:
Advanced Server	`enterprisedb`
PostgreSQL	`postgres` or `postgresql`

`schema_name`

`schema_name` is the name of the schema within the source database (specified in the `toolkit.properties` file) that you wish to migrate. You must include at least one *`schema_name`*.

The following example migrates a schema (table definitions and table content) named HR from a MySQL database on a Linux system to an Advanced Server host. Note that the command includes the -sourcedbtype and targetdbtype options:

```
$ ./runMTK.sh -sourcedbtype mysql -targetdbtype
enterprisedb HR
```

On Windows, use the following command:

```
> .\runMTK.bat -sourcedbtype mysql -targetdbtype
enterprisedb HR
```

You can migrate multiple schemas from a source database by including a comma-delimited list of schemas at the end of the Migration Toolkit command. The following example migrates multiple schemas (named HR and ACCTG) from a MySQL database to a PostgreSQL database:

On Linux, use the following command to migrate multiple schemas from a MySQL database:

```
    $ ./runMTK.sh -sourcedbtype mysql -targetdbtype postgres
HR,ACCTG
```

On Windows, use the following command form:

```
    > .\runMTK.bat -sourcedbtype mysql -targetdbtype postgres
HR,ACCTG
```


7 Migration Toolkit Command Options

Append migration options when you run Migration Toolkit to conveniently control details of the migration. For example, to migrate all schemas within a database, append the `-allSchemas` option to the command:

```
$ ./runMTK.sh -allSchemas
```

Sections 7.1 through 7.7 of this document contain reference material for each of the command options that work with Migration Toolkit; options are grouped by their behavior. The table below shows the section number (within this document) that describes each set of features.

Feature:	Section	Relevant Options:
Offline Migration	7.1	`-offlineMigration`
Import Options	7.2	`-sourcedbtype, -targetdbtype, -schemaOnly, -dataOnly`
Schema Creation Options	7.3	`-dropSchema, -targetSchema`
Schema Object Selection Options	7.4	`-allTables, -tables, -importPartitionAsTable, -constraints, -ignoreCheckConstFilter, -skipCKConst, -skipFKConst, -skipColDefaultClause, -indexes, -triggers, -allViews, -views, -allSequences, -sequences, -allProcs, -procs, -allFuncs, -funcs, -checkFunctionBodies, -allPackages, -packages, -allRules,`
Migration Options	7.5	`-truncLoad, -enableConstBeforeDataLoad, -retryCount, -safeMode, -fastCopy, -analyze, vacuumAnalyze, -replaceNullChar, -copyDelimiter, -batchSize, -cpBatchSize, -fetchSize, -filterProp -customColTypeMapping, -customColTypeMappingFile`
Oracle Specific Options	7.6	`-allUsers, -users, -objectTypes, -copyViaDBLinkOra, -allDBLinks -allSynonyms, -allPublicSynonyms, -allPrivateSynonyms`
Miscellaneous Options	7.7	`-help, -logDir, -logFileCount, -logFileSize, -verbose, -version`

7.1 Offline Migration

If you specify the `-offlineMigration` option in the command line, Migration Toolkit performs an *offline* migration. During an offline migration, Migration Toolkit reads the definition of each selected object and creates an SQL script that, when executed at a later time, replicates each object in Postgres.

Note: The following examples demonstrate invoking Migration Toolkit in Linux; to invoke Migration Toolkit in Windows, substitute the `runMTK.bat` command for the `runMTK.sh` command.

To perform an offline migration of both schema and data, specify the `-offlineMigration` keyword, followed by the schema name:

```
$ ./runMTK.sh -offlineMigration schema_name
```

Each database object definition is saved in a separate file with a name derived from the schema name and object type in the user's home folder. To specify an alternative file destination, include a directory name after the `-offlineMigration` option:

```
$ ./runMTK.sh -offlineMigration file_dest schema_name
```

To perform an offline migration of schema objects only (creating empty tables), specify the `-schemaOnly` keyword in addition to the `-offlineMigration` keyword when invoking Migration Toolkit:

```
$ ./runMTK.sh -offlineMigration -schemaOnly schema_name
```

To perform an offline migration of data only (omitting any schema object definitions), specify the `-dataOnly` keyword and the `-offlineMigration` keyword when invoking Migration Toolkit

```
$ ./runMTK.sh -offlineMigration -dataOnly schema_name
```

By default, data is written in `COPY` format; to write the data in a plain SQL format, include the `-safeMode` keyword:

```
$ ./runMTK.sh -offlineMigration -dataOnly -safeMode
schema_name
```

By default, when you perform an offline migration that contains table data, a separate file is created for each table. To create a single file that contains the data from multiple tables, specify the `-singleDataFile` keyword:

```
./runMTK.sh -offlineMigration -dataOnly -singleDataFile -
safeMode schema_name
```


Please note: the -singleDataFile option is available only when migrating data in a plain SQL format; you must include the -safeMode keyword if you include the -singleDataFile option.

7.1.1 Executing Offline Migration Scripts

You can use the edb-psql command line (on Advanced Server) or psql command line (on PostgreSQL) to execute the scripts generated during an offline migration. The following example describes restoring a schema (named hr) into a new database (named acctg) stored in Advanced Server.

1. Use the createdb utility to create the acctg database, into which we will restore the migrated database objects:

   ```
   createdb -U enterprisedb acctg
   ```

2. Connect to the new database with edb-psql:

   ```
   edb-psql -U enterprisedb acctg
   ```

3. Use the \i meta-command to invoke the migration script that creates the object definitions:

   ```
   acctg=# \i ./mtk_hr_ddl.sql
   ```

4. If the -offlineMigration command included the -singleDataFile keyword , the mtk_hr_data.sql script will contain the commands required to recreate all of the objects in the new target database. Populate the database with the command:

   ```
   acctg=# \i ./mtk_hr_data.sql
   ```


7.2 Import Options

By default, Migration Toolkit assumes the source database to be Oracle and the target database to be Advanced Server; include the `-sourcedbtype` and `-targetdbtype` keywords to specify a non-default source or target database.

By default, Migration Toolkit imports both the data and the object definition when migrating a schema; alternatively you can choose to import either the data or the object definitions.

`-sourcedbtype` *`source_type`*

> The `-sourcedbtype` option specifies the source database type. *`source_type`* may be one of the following values: `mysql`, `oracle`, `sqlserver`, `sybase`, `postgresql` or `enterprisedb`. *`source_type`* is case-insensitive. By default, *`source_type`* is `oracle`.

`-targetdbtype` *`target_type`*

> The `-targetdbtype` option specifies the target database type. *`target_type`* may be one of the following values: `enterprisedb`, `postgres`, or `postgresql`. *`target_type`* is case-insensitive. By default, *`target_type`* is `enterprisedb`.

`-schemaOnly`

> This option imports the schema definition and creates all selected schema objects in the target database. This option cannot be used in conjunction with `-dataOnly` option.

`-dataOnly`

> This option copies the table data only. When used with the `-tables` option, Migration Toolkit will only import data for the selected tables (see usage details below). This option cannot be used with `-schemaOnly` option.

7.3 Schema Creation Options

By default, Migration Toolkit imports the source schema objects and/or data into a schema of the same name. If the target schema does not exist, Migration Toolkit creates a new schema. Alternatively, you may specify a custom schema name via the `-targetSchema` option. You can choose to drop the existing schema and create a new schema using the following option:

`-dropSchema [true|false]`

> When set to `true`, Migration Toolkit drops the existing schema (and any objects within that schema) and creates a new schema. (By default, `-dropSchema` is `false`).

`-targetSchema` *`schema_name`*

> Use the `-targetSchema` option to specify the name of the migrated schema. If you are migrating multiple schemas, specify a name for each schema in a comma-separated list. If the command line does not include the `-targetSchema` option, the name of the new schema will be the same as the name of the source schema.

You cannot specify `information-schema`, `dbo`, `sys` or `pg_catalog` as target schema names. These schema names are reserved for meta-data storage in Advanced Server.

7.4 Schema Object Selection Options

Use the following options to select specific schema objects to migrate:

`-allTables`

> Import all tables from the source schema.

`-tables` *`table_list`*

> Import the selected tables from the source schema. *`table_list`* is a comma-separated list of table names (e.g. `-tables emp, dept, acctg`).

`-importPartitionAsTable` *`table_list`*

> Include the `-importPartitionAsTable` parameter to import the contents of a partitioned table into a single non-partitioned table. *`table_list`* is a comma-separated list of table names (e.g. `-importPartitionAsTable emp, dept, acctg`).

`-constraints`

Import the table constraints. This option is valid only when importing an entire schema or when the `-allTables` or `-tables` *`table_list`* options are specified.

`-ignoreCheckConstFilter`

By default, Migration Toolkit does not implement migration of check constraints and default clauses from a Sybase database. Include the `-ignoreCheckConstFilter` parameter when specifying the `-constraints` parameter to migrate constraints and default clauses from a Sybase database.

`-skipCKConst`

Omit the migration of check constraints. This option is useful when migrating check constraints that are based on built-in functions (in the source database) that are not supported in the target database.

This option is valid only when importing an entire schema or when the `-allTables` or `-tables` *`table_list`* options are specified.

`-skipFKConst`

Omit the migration of foreign key constraints. This option is valid only when importing an entire schema or when the `-allTables` or `-tables` *`table_list`* options are specified.

`-skipColDefaultClause`

Omit the migration of the column `DEFAULT` clause.

`-indexes`

Import the table indexes. This option is valid when importing an entire schema or when the `-allTables` or `-tables` *`table_list`* option is specified.

`-triggers`

Import the table triggers. This option is valid when importing an entire schema or when the `-allTables` or `-tables` *`table_list`* option is specified.

`-allViews`

Import the views from the source schema. Please note that this option will migrate both dynamic *and* materialized views from the source.

`-views` *`view_list`*

Import the specified materialized or dynamic views from the source schema. *`view_list`* is a comma-separated list of view names (e.g. `-views all_emp,mgmt_list,acct_list`)

`-allSequences`

Import all sequences from the source schema.

`-sequences` *`sequence_list`*

Import the selected sequences from the source schema. *`sequence_list`* is a comma-separated list of sequence names.

`-allProcs`

Import all stored procedures from the source schema.

`-procs` *`procedures_list`*

Import the selected stored procedures from the source schema. *`procedures_list`* is a comma-separated list of procedure names.

`-allFuncs`

Import all functions from the source schema.

`-funcs` *`function_list`*

Import the selected functions from the source schema. *`function_list`* is a comma-separated list of function names.

`-checkFunctionBodies [true/false]`

When `false`, disables validation of the function body during function creation (to avoid errors if the function contains forward references). The default value is `true`.

`-allPackages`

Import all packages from the source schema.

`-packages` *`package_list`*

Import the selected packages from the source schema. *`package_list`* is a comma-separated list of package names.

`-allRules`

Import all rules from the source database; this option is only valid when both the source and target are stored on a Postgres host.

7.5 Migration Options

Use the migration options listed below to control the details of the migration process.

`-loaderCount [value]`

Use the `-loaderCount` option to specify the number of parallel threads that Migration Toolkit should use when importing data. This option is particularly useful if the source database contains a large volume of data, and the Postgres host (that is running Migration Toolkit) has high-end CPU and RAM resources. While *`value`* may be any non-zero, positive number, we recommend that *`value`* should not exceed the number of CPU cores; a dual core CPU should have an optimal *`value`* of `2`.

Please note that specifying too large of a *`value`* could cause Migration Toolkit to terminate, generating a `'Out of heap space'` error.

`-truncLoad`

Truncate the data from the table before importing new data. This option can only be used in conjunction with the `-dataOnly` option.

`-enableConstBeforeDataLoad`

Include the `-enableConstBeforeDataLoad` option if a non-partitioned source table is mapped to a partitioned table. This option enables all triggers on the target table (including any triggers that redirect data to individual partitions) before the data migration. `-enableConstBeforeDataLoad` is valid only if the `-truncLoad` parameter is also specified.

`-retryCount [value]`

If you are performing a multiple-schema migration, objects that fail to migrate during the first migration attempt due to cross-schema dependencies may successfully migrate during a subsequent migration. Use the `-retryCount` option to specify the number of attempts that Migration Toolkit will make to migrate an object that has failed during an initial migration attempt. Specify a *`value`* that is greater than `0`; the default value is `2`.

`-safeMode`

If you include the `-safeMode` option, Migration Toolkit commits each row as migrated; if the migration fails to transfer all records, rows inserted prior to the point of failure will remain in the target database.

`-fastCopy`

Including the `-fastCopy` option specifies that Migration Toolkit should bypass WAL logging to perform the `COPY` operation in an optimized way, default disabled. If you choose to use the `-fastCopy` option, migrated data may not be recoverable (in the target database) if the migration is interrupted.

`-replaceNullChar` *value*

The Migration Toolkit properly supports importation of a column where its value is `NULL`.

However, the Migration Toolkit does not support importation of `NULL` character values (embedded binary zeros 0x00) with the JDBC connection protocol. If you are importing data that includes the `NULL` character, use the `-replaceNullChar` option to replace the `NULL` character with a single, non-`NULL`, replacement character. Do not enclose the replacement character in quotes or apostrophes.

Once the data has been migrated, use a SQL statement to replace the character specified by `-replaceNullChar` with binary zeros.

`-analyze`

Include the `-analyze` option to invoke the Postgres `ANALYZE` operation against a target database. The optimizer consults the statistics collected by the `ANALYZE` operation, utilizing the information to construct efficient query plans.

`-vacuumAnalyze`

Include the `-vacuumAnalyze` option to invoke both the `VACUUM` and `ANALYZE` operations against a target database. The optimizer consults the statistics collected by the `ANALYZE` operation, utilizing the information to construct efficient query plans. The `VACUUM` operation reclaims any storage space occupied by dead tuples in the target database.

`-copyDelimiter`

Specify a single character to be used as a delimiter in the copy command when loading table data. The default value is '`\t`' (`tab`).

`-batchSize`

Specify the batch size of bulk inserts. Valid values are 1-1000. The default batch size is 1000; reduce the value of `-batchSize` if `Out of Memory` exceptions occur.

`-cpBatchSize`

Specify the batch Size in MB, to be used in the `COPY` command. Any value greater than 0 is valid; the default batch size is 8 MB.

`-fetchSize`

Use the `-fetchSize` option to specify the number of rows fetched in a result set. If the designated `-fetchSize` is too large, you may encounter `Out of Memory` exceptions; include the `-fetchSize` option to avoid this pitfall when migrating large tables. The default fetch size is specific to the JDBC driver implementation, and varies by database.

MySQL users note: By default, the MySQL JDBC driver will fetch all of the rows that reside in a table into the client application (Migration Toolkit) in a single network round-trip. This behavior can easily exceed available memory for large tables. If you encounter an `'out of heap space'` error, specify `-fetchSize 1` as a command line argument to force Migration Toolkit to load the table data one row at a time.

`-filterProp` *`file_name`*

`file_name` specifies the name of a file that contains constraints in `key=value` pairs. Each record read from the database is evaluated against the constraints; those that satisfy the constraints are migrated. The left side of the pair lists a table name; please note that the table name should *not* be schema-qualified. The right side specifies a condition that must be true for each row migrated. For example, including the following constraints in the property file:

```
countries=country_id<>'AR'
```

migrates only those countries with a `country_id` value that is not equal to `AR`; this constraint applies to the `countries` table.

`-customColTypeMapping` *`column_list`*

Use custom type mapping to change the data type of migrated columns. The left side of each pair specifies the columns with a regular expression; the right side of each pair names the data type that column should assume. You can include multiple pairs in a semi-colon separated *`column_list`*. For example, to map any

column whose name ends in `ID` to type `INTEGER`, use the following custom mapping entry:

```
.*ID=INTEGER
```

Custom mapping is applied to all table columns that match the criteria unless the column is table-qualified.

The '`\\`' characters act as an escape string; since '`.`' is a reserved character in regular expressions, use '`\\.`' to represent the '`.`' character. For example, to use custom mapping to select rows from the `EMP_ID` column in the `EMP` table, specify the following custom mapping entry:

```
EMP\\.EMP_ID=INTEGER
```

`-customColTypeMappingFile property_file`

You can include multiple custom type mappings in a `property_file`; specify each entry in the file on a separate line, in a `key=value` pair. The left side of each pair selects the columns with a regular expression; the right side of each pair names the data type that column should assume.

7.6 Oracle Specific Options

The following options apply only when the source database is Oracle.

`-objecttypes` *`schema_name`*

Import the user-defined object types from the specified schema.

`-allUsers`

Import all users and roles from the source database. Please note that the `-allusers` option is only supported when migrating from an Oracle database to an Advanced Server database.

`-users` *`user_list`*

Import the selected users or roles from the source Oracle database. *`user_list`* is a comma-separated list of user/role names (e.g. `-users MTK, SAMPLE, acctg`). Please note that the `-users` option is only supported when migrating from an Oracle database to an Advanced Server database.

`-copyViaDBLinkOra`

The `dblink_ora` module provides Advanced Server-to-Oracle connectivity at the SQL level. `dblink_ora` is bundled and installed as part of the Advanced Server database installation. `dblink_ora` utilizes the `COPY` API method to transfer data between databases. This method is considerably faster than the `JDBC COPY` method.

The following example uses the `dblink_ora COPY` API to migrate all tables from the `HR` schema:

```
$./runMTK.sh -copyViaDBLinkOra -allTables HR
```

The target Advanced Server database must have `dblink_ora` installed and configured. For installation details, refer to the `dblink_ora` setup guide (`README-DBLINK_ORA_SETUP.txt`), in the `doc/contrib` subdirectory under the Postgres Plus Advanced Server installation home directory.

`-allDBLinks [link_Name_1=password_1,link_Name_2=password_2,...]`

Choose this option to migrate Oracle database links. The password information for each link connection in the source database is encrypted, so unless specified, a dummy password (`edb`) is substituted.

To migrate all database links using `edb` as the dummy password for the connected user:

```
$./runMTK.sh -allDBLinks HR
```

You can alternatively specify the password for each of the database links through a comma-separated list of `name=value` pairs. Specify the link name on the left side of the pair and the password value on the right side.

To migrate all database links with the actual passwords specified on the command-line:

```
$./runMTK.sh -allDBLinks LINK_NAME1=abc,LINK_NAME2=xyz
HR
```

Migration Toolkit migrates only the database link types that are currently supported by EnterpriseDB; this includes fixed user links of public and private type.

`-allSynonyms`

Include the `-allSynonyms` option to migrate all public and private synonyms from an Oracle database to an Advanced Server database. If a synonym with the same name already exists in the target database, the existing synonym will be replaced with the migrated version.

`-allPublicSynonyms`

Include the `-allPublicSynonyms` option to migrate all public synonyms from an Oracle database to an Advanced Server database. If a synonym with the same name already exists in the target database, the existing synonym will be replaced with the migrated version.

`-allPrivateSynonyms`

Include the `-allPrivateSynonyms` option to migrate all private synonyms from an Oracle database to an Advanced Server database. If a synonym with the same name already exists in the target database, the existing synonym will be replaced with the migrated version.

7.7 Miscellaneous Options

Use the migration options listed below to view Migration Toolkit help and version information; you can also use the options in this section to control Migration Toolkit feedback and logging options.

`-help`

Display the application command-line usage information.

`-logDir` *`log_path`*

Include this option to specify where the log files will be written; *`log_path`* represents the path where application log files are saved. By default, on Linux log files are written to:

```
$HOME/.enterprisedb/migration-toolkit/logs
```

On Windows, the log files are saved to:

```
%HOMEDRIVE%%HOMEPATH%\.enterprisedb\migration-
toolkit\logs
```


`-logFileCount` *`file_count`*

Include this option to specify the number of files used in log file rotation. Specify a value of `0` to disable log file rotation and create a single log file (it will be truncated when it reaches the value specified using the `logFileSize` option). *`file_count`* must be greater than or equal to `0`; the default is 20.

`-logFileSize` *`file_size`*

Include this option to specify the maximum file size limit (in MB) before rotating to a new log file. *`file_size`* must be greater than `0`; the default is 50 MB.

`-verbose [on|off]`

Display application log messages on standard output (By default, verbose is `on`).

`-version`

Display the Migration Toolkit version.

8 Migration Issues

During the migration process, the migration summary displays the progress of the migration. If an error occurs, the summary will display information about the error. The migration summary is also written to a log file. The default locations for the log files are:

On Linux:

```
$HOME/.enterprisedb/migration-toolkit/logs
```

On Windows, the log files are saved to:

```
%HOMEDRIVE%%HOMEPATH%\.enterprisedb\migration-
toolkit\logs
```

You can specify an alternate log file directory with the `-logdir` *`log_path`* option in Migration Toolkit.

8.1 Migration Toolkit Connection Errors

Migration Toolkit uses information from the `toolkit.properties` file to connect to the source and target databases. Most of the connection errors that occur when using Migration Toolkit are related to the information specified in the `toolkit.properties` file. Use the following section to identify common connection errors, and learn how to resolve them.

For information about editing the `toolkit.properties` file, see Section 5, *Building the toolkit.properties file*.

8.1.1 Invalid username/password

When I try to perform a migration from an Oracle database with Migration Toolkit, I get the following error:

```
Error: java.lang.Exception: ORA-01017: invalid username/password; logon
denied.
```

The user name or password specified in the `toolkit.properties` file is not valid to use to connect to the Oracle source database.

To resolve this error, edit the `toolkit.properties` file, specifying the name and password of a valid user with sufficient privileges to perform the migration in the `SRC_DB_USER` and `SRC_DB_PASSWORD` properties.

8.1.2 Connection rejected: FATAL: password

When I try to perform a migration with Migration Toolkit, I get the following error:

```
Error: java.lang.Exception: Connection rejected: FATAL: password
authentication failed for user "name".
```

The user name or password specified in the `toolkit.properties` file is not valid to use to connect to the Postgres database.

To resolve this error, edit the `toolkit.properties` file, specifying the name and password of a valid user with sufficient privileges to perform the migration in the `TARGET_DB_USER` and `TARGET_DB_PASSWORD` properties.

8.1.3 Exception: ORA-28000: the account is locked

When I try to perform a migration from an Oracle database with Migration Toolkit, I get the following error message:

```
Error: java.lang.Exception: ORA-28000: the account is locked.
```

The Oracle account associated with the user name specified in the `toolkit.properties` file is locked.

To resolve this error, you can either unlock the user account on the Oracle server or edit the `toolkit.properties` file, specifying the name and password of a valid user with sufficient privileges to perform the migration in the `SRC_DB_USER` and `SRC_DB_PASSWORD` parameters.

8.1.4 Exception: oracle.jdbs.driver.OracleDriver

When I try to perform a migration with Migration Toolkit, the migration fails and I get the error message:

```
Error: java.lang.Exception: oracle.jdbs.driver.OracleDriver
```

Before using Migration Toolkit, you must download and install the appropriate JDBC driver for the database that you are migrating from. See Section 4.3, *Installing Source-Specific Drivers* for complete instructions.

8.1.5 I/O exception: The Network Adapter could not establish the connection

When I try to perform a migration with Migration Toolkit, I get the following error:

```
Error: java.lang.Exception: I/O exception: The Network Adapter could not
establish the connection
```

The JDBC URL for the source database specified in the `toolkit.properties` file contains invalid connection properties.

To resolve this error, edit the `toolkit.properties` file, specifying valid connection information for the source database in the `SRC_DB_URL` property. For information about forming a JDBC URL for your specific database, see Sections 5.1 through 5.6 of this document.

8.1.6 Exception: The connection attempt failed

When I try to perform a migration with Migration Toolkit, I get the following error:

```
Error: java.lang.Exception: The connection attempt failed.
```

The JDBC URL for the target database (Advanced Server) specified in the `toolkit.properties` file contains invalid connection properties.

To resolve this error, edit the `toolkit.properties` file, specifying valid connection information for the target database in the `TARGET_DB_URL` property. For information about forming a JDBC URL for Advanced Server, see section 5.1 of this document.

8.2 Migration Toolkit Migration Errors

The following errors may occur after Migration Toolkit has successfully connected to the target and source database servers.

8.2.1 ERROR: Extra Data after last expected column

When migrating a table online, I get the error message:

```
Error Loading Data into Table: column_name: ERROR: extra data after last
expected column
```

This error occurs when the data in the column *column_name* includes the delimiter character. To correct this error, change the delimiter character to a character not found in the table contents.

8.2.2 Error Loading Data into Table: *TABLE_NAME*: null

When performing a data-only migration, I get the following error:

```
Trying to reload table: TABLE_NAME through bulk inserts with a batch size of
1000
Error Loading Data into Table: TABLE_NAME: null
Data Load Summary: Total Time (sec): 0.0 Total Rows: 0 Total Size(MB): 0.0

Schema HR imported with errors.
```

You must create a table to receive the data in the target database before you can migrate the data. Verify that a table (with a name of *TABLE_NAME*) exists in the target database; create the table if necessary and re-try the data migration.

8.2.3 Error Creating Constraint CONS_NAME_FK

When I perform a table migration that includes indexes and constraints, the log file includes the following error message:

```
Error Creating Constraint EMP_DEPT_FK: ERROR: relation "hr.departments" does
not exist

Schema HR imported with errors.
One or more schema objects could not be imported during the migration
process. Please review the migration output for more details.

Migration logs have been saved to C:\Docume~1\susan\.enterprisedb\migration-
toolkit\logs

******************** Migration Summary ********************
Tables: 1 out of 1
Constraints: 5 out of 6
Total objects: 11
Successful count: 10
Failure count: 1
```

```
List of failed objects
======================
Constraints
-------------------
1.  HR.EMPLOYEES.EMP_DEPT_FK
```

The table you are migrating includes a foreign key constraint on a table that does not exist in the target database. Migration Toolkit creates the table, omitting the foreign key constraint.

You can avoid generating the error message by including the `-skipFKConst` option in the Migration Toolkit command.

8.2.4 Error Loading Data into Table

I've already migrated the table definition; when I try to migrate the data into the table, I get an error:

```
Error Loading Data into Table: DEPARTMENTS: ERROR: extra data after last
expected column
Where: COPY departments, line 1: "10  Administration      200    1700"
Trying to reload table: DEPARTMENTS through bulk inserts with a batch size of
1000

Batch entry 0 INSERT INTO hr.DEPARTMENTS VALUES ('10', 'Administration',
'200', '1700'); was aborted.  Call getNextException to see the cause.,
Skipping Batch
Stack Trace:
java.sql.BatchUpdateException: Batch entry 0 INSERT INTO hr.DEPARTMENTS
VALUES ('10', 'Administration', '200', '1700'); was aborted.

Table Data Load Summary: Total Time(s): 0.235 Total Rows: 0
Data Load Summary: Total Time (sec): 0.235 Total Rows: 0 Total Size(MB): 0.0

Schema HR imported with errors.
```

The table definition (in the target database) does not match the migrated data. If you've altered the target or source table, confirm that the table definition and the data are compatible.

8.2.5 ERROR: value too long for type

I've already migrated the table definition; when I try to migrate the data into the table, I get the following error:

```
Error Loading Data into Table: DEPARTMENTS: ERROR: value too long for type
character(1)
  Where: COPY departments, line 1, column location_id: "1700"
Trying to reload table: DEPARTMENTS through bulk inserts with a batch size of
1000
Batch entry 0 INSERT INTO hr.DEPARTMENTS VALUES ('10', 'Administration',
'200', '1700'); was aborted.
```

A column in the target database is not large enough to receive the migrated data; this problem could occur if the table definition is altered after migration. The column name (in our example, `location_id`) is identified in the line that begins with '`Where`:'.

```
Where: COPY departments, line 1, column location_id: "1700"
```

To correct the problem, adjust the column size and re-try the migration.

8.2.6 ERROR: Exception on Toolkit thread:OutOfMemoryError

When migrating from a MySQL database, I encounter the following error:

```
Loading Table: doc_views…
Exception on Toolkit thread: java.lang.OutOfMemoryError: Java heap space
java.lang.OurOfMemoryError: Java heap space
at
java.lang.AbstracStringBuilder.expandCapacity(AbstractStringBuilder?.java:99)
```

By default, the MySQL JDBC driver will fetch all of the rows that reside in a table into the client application (Migration Toolkit) in a single network round-trip. This behavior can easily exceed available memory for large tables.

To correct the problem, specify `-fetchSize 1` as a command line argument when you re-try the migration.

8.3 Unsupported Oracle Features

Advanced Server offers complete support for some Oracle features and partial support for others. Migration Toolkit cannot migrate any object that uses an unsupported feature.

In some cases, Migration Toolkit can migrate objects that use features that offer partial compatibility. In other cases, Advanced Server supports suitable workarounds.

Full-text search is an example of functionality that is not fully compatible with Oracle. The Advanced Server database has included support for full-text search for quite some time, but the implementation is quite different than Oracle's; Migration Toolkit is unable to migrate objects that utilize this feature.

There are also features that Advanced Server does not yet support. Features in this category include Automated Storage Management, Advanced Queuing, table compression, and external tables. You can often orchestrate a successful workaround:

- Automated Storage Management can be replaced with (system specific) volume management software.
- Advanced Queuing can be replaced by external messaging systems such as ActiveMQ, TIBCO or MQ Series.
- Table compression can be implemented by storing data in a tablespace that resides on a compressed filesystem.
- External tables don't exist in Advanced Server, but you *can* load flat text files into staging tables in the database. We recommend using the EDB*Loader utility to load the data into an Advanced Server database quickly.

8.4 Frequently Asked Questions

Does Migration Toolkit support the migration of packages?

Migration Toolkit supports the migration of packages from an Oracle database into Advanced Server. See Section 3, *Functionality Overview* for information about the migration support offered by Advanced Server.

Is there a way to transfer only the data from the source database?

Yes. Data transfer is supported as part of an online or offline migration.

Does Migration Toolkit support migration of tables that contain data of the CLOB data type?

Migration Toolkit does support migration of tables containing data of the `CLOB` type.

Does Advanced Server support the enum data type?

Advanced Server does not currently support the `enum` data type, but will support them in future releases. Until then, you can use a check constraint to restrict the data added to an Advanced Server database. A check constraint defines a list of valid values that a column may take.

The following code sample includes a simple example of a check constraint that restricts the value of a column to one of three `dept` types; `sales`, `admin` or `technical`.

```
CREATE TABLE emp (
  emp_id INT NOT NULL PRIMARY KEY,
  dept VARCHAR(255) NOT NULL,

  CHECK (dept IN ('sales', 'admin', 'technical'))
);
```

If we test the check constraint by entering a valid `dept` type, the `INSERT` statement works without error:

```
test=# INSERT INTO emp VALUES (7324, 'sales');
INSERT 0 1
```

If we try to insert a value not included in the constraint (`support`), Advanced Server throws an error:

```
test=# INSERT INTO emp VALUES (7325, 'support');
ERROR: new row for relation "emp" violates check constraint "emp_dept_check"
```

Does Advanced Server support materialized views?

Postgres does not support Oracle compatible materialized views. To setup a materialized view/summary table in Postgres you must manually create the triggers that maintain the summary table. Automatic query rewrite is not currently supported; the application must be made aware of the summary table's existence.

When I try to migrate from a MySQL database that includes a TIME data type, I get the following error: Error Loading Data into Table: Bad format for Time. Does Postgres support MySQL TIME data types?

Postgres will have no problem storing `TIME` data types as long as the value of the hour component is not greater than `24`.

Unlike Postgres, the MySQL `TIME` data type will allow you to store a value that represents either a `TIME` or an `INTERVAL` value. A value stored in a MySQL `TIME` column that represents an `INTERVAL` value could potentially be out of the accepted range of a valid Postgres `TIMESTAMP` value. If, during the migration process, Postgres encounters a value stored in a `TIME` data column that it perceives as out of range, it will return an error.