SelExL

Installation and Quick Start Guide

SelExL 1.2
Installation and Quick Start Guide

SelExL by Christallize

Manual Edition 1 – December 2006

Christallize Ltd

2, Carvel Way, Littlehampton, West Sussex, BN17 6RJ

United Kingdom

Sales

+44 1903 714200

Home page: www.christallize.com

E-mail: sales@christallize.com

Table of Contents

About This Guide

The SelExL installation and quick-start guide is provided to help you get up and running with SelExL as quickly as possible. This is accomplished with the use of a small sample database schema.

The installation and quick start guide consists of the following sections:

- **Chapter 1: Introduction** provides a brief overview of SelExL, the reason for subset databases and the purpose behind the sample schema.
- **Chapter 2: Installation** lists the system requirements for SelExL, the required installation steps and what you need to be able to run SelExL against the sample schema.
- **Chapter 3: Setting up the Sample Schema** covers the necessary steps to set-up and install SelExL's sample schema in a database.
- **Chapter 4: Configuration** explains the process of building SelExL's required configuration files.
- **Chapter 5: Running SelExL** describes how to run SelExL.
- **Appendix A: Sample Schema** describes the sample schema with use of a basic Entity-Relationship diagram.
- **Appendix B: Step-by-Step Configuration** provides a step-by-step guide on how to produce SelExL's required configuration files for the sample schema.

Whilst reading this guide you will come across new terminology that helps to explain and define SelExL. When the new terminology is first introduced it will be highlighted(1) with a footnote to provide a full explanation.

The following typographical conventions are used in this manual:

Italic

Used for filenames, directory names and for emphasis.

`Constant width`

Used for instructions that need entering at the command line and listing example file contents.

In a number of examples file and directory names are included. Sometimes this will be using Windows file names and sometimes UNIX style file names. In all cases the style of file name can be inter-changed to fit the platform that SelExL is running on.

If you have any comments, corrections or questions regarding this manual then please Email them to feedback@christallize.com

(1) Highlighted – Highlighted terminology explained in a footnote.

CHAPTER 1

Introduction

This chapter includes the following information:

- Why have Database Subsets?
- How SelExL works
- Using the Sample Schema with SelExL

1 INTRODUCTION

SelExL for Oracle is designed to create a referentially correct data subset from an Oracle database and use this subset to populate an empty, smaller copy database (a “subset database”). Thus isolating a selected portion of the original database into a separate database for more manageable testing and troubleshooting.

1.1 Why have Database Subsets?

Databases today are growing in both size and complexity to meet the ever-increasing demands of business. Applications to process the data in these databases are also increasing in size and complexity. Alongside this, organisations want rapid turn around for changes to these systems. When a database exceeds a certain size it becomes very expensive (in terms of elapsed time, hardware and manpower) to provide full-size copies of the production database for development, testing and training.

One resolution to this problem is to have fewer full size copies of the production database than are really needed, commonly only one, and ask the various development and testing teams to share.

Clearly this is far from optimal. A great deal of effort is required to manage and schedule usage of the production copy. Data in the database is left in an unknown state when passed from one team to the next. It takes a long time, frequently more than a day, to provide a refresh of the production copy when it’s required.

The reality is that databases required by training and application development teams rarely need to be full size, performance and volume testing are possible exceptions. In fact it is often easier to work on a small copy as response times and batch runtimes are much quicker. What is often required is a smaller version of the production database that correctly replicates the database structure and content of the larger database.

Creating a smaller, referentially correct subset database provides great benefits to the IT department and the business as a whole, although it is not necessarily a simple task. However the quicker, easier and more automated the process is, the better it is for everyone. This is where SelExL becomes an indispensable tool.

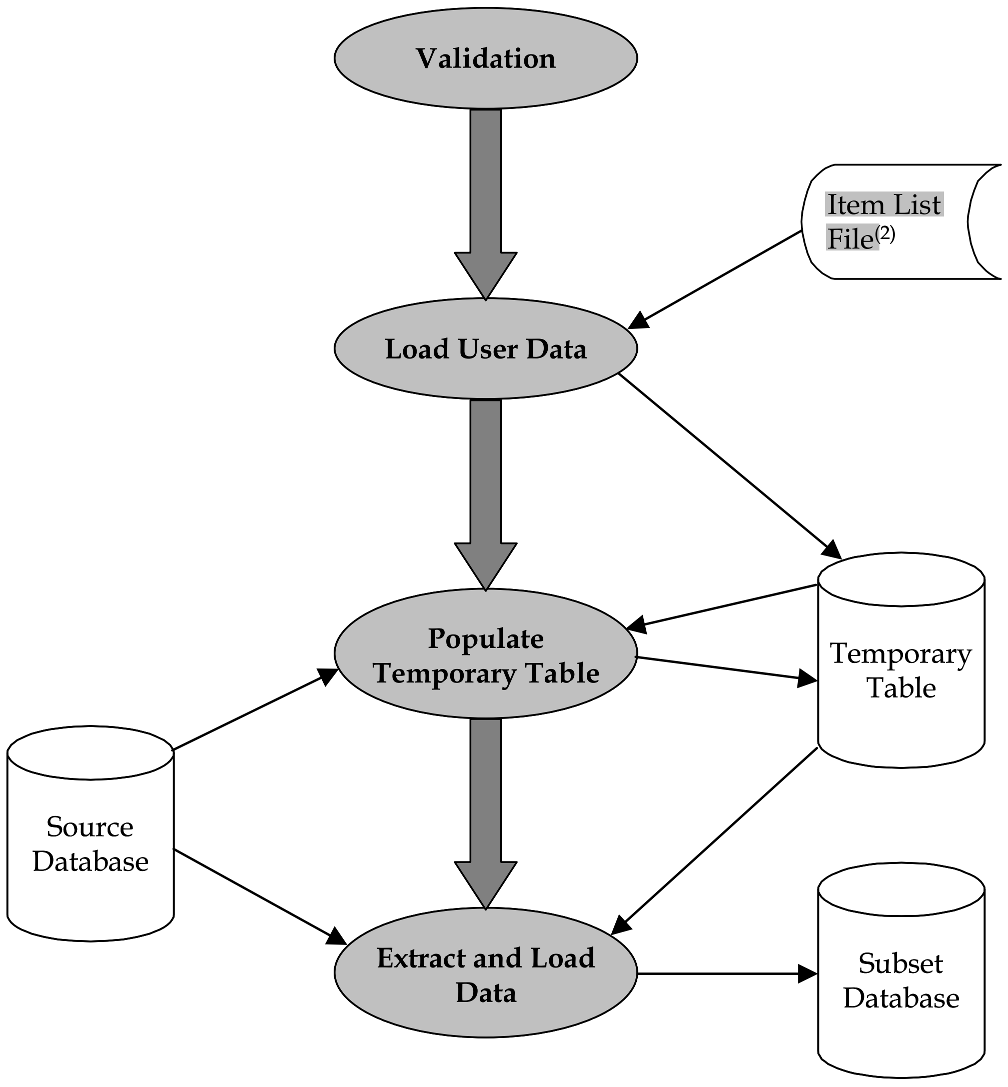
1.2 How SelExL works

SelExL is designed to run on any Windows, Linux or UNIX machine with an Oracle client installation. SelExL can be used against any Oracle database on any platform through the use of Oracle Net, (NET8 for Oracle 8i).

SelExL does not require any database enforced referential integrity to exist in the source database(1) i.e. primary key and foreign key constraints do not need to exist.

Once the initial setup and configuration tasks are completed SelExL is ready to run. SelExL can run in two modes. Firstly by extracting and loading the selected data into the target subset database as a single process (EXLO, EXtract and LOad). Secondly by extracting all the data to files (BEX, Bulk EXtract) and subsequently loading from these files into the target subset database (BLO, Bulk LOad).

Combined Extract and Load (EXLO)



(1) Source database – The database from where data is extracted to populate the subset database, usually the production database.

(2) Item list file – A file with a list of user defined values to determine the extract. This is likely to change each time SelExL performs an extraction. E.g. in a healthcare system this might be a list of patient ids.

Validation - SelExL validates all the configuration files to ensure all entries are syntactically correct and consistent with entries in related files.

Load User Data - SelExL inserts the user supplied list of values for Extract Key[1], from the item list file, into a temporary table, EXTRACT_KEYS. For example a list of account numbers, employee ids. (In this case Extract Key is account_id and employee_id respectively.) If this list contains wildcards then these are used to obtain values for the Extract Key from the Driving Table[2]. Suppose you wished to extract a random 1% sample of bank accounts from the production database. The user supplied value contained in the item list file could be *%01*.

Populate Temporary Table – Complete populating EXTRACT_KEYS. SelExL uses information from the configuration files to populate EXTRACT_KEYS with all the required key values for each table in the extract.

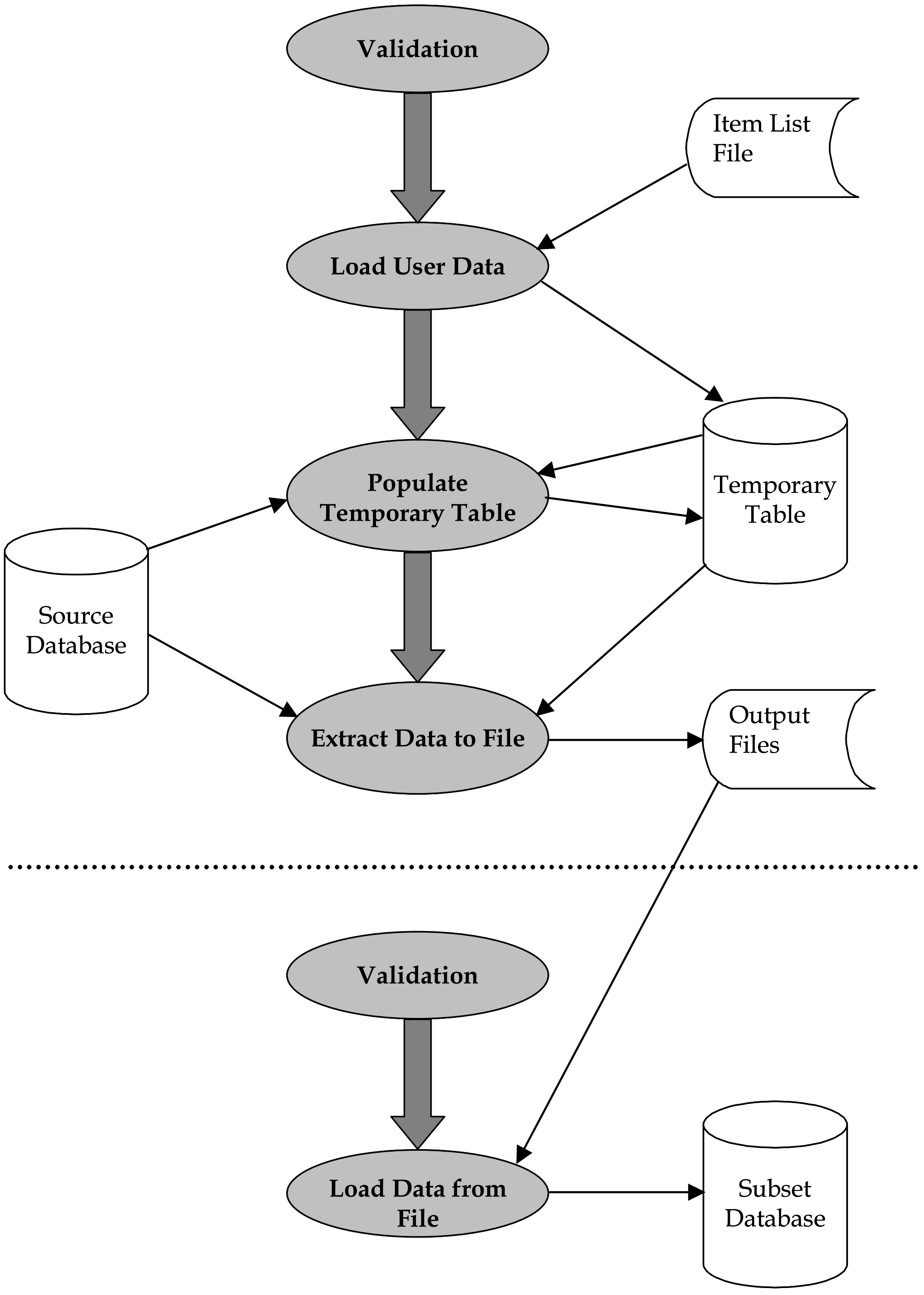
Extract and Load Data - SelExL extracts the required rows for each table using the keys from EXTRACT_KEYS and then inserts the data into the target subset database. This is carried out using multiple streams.

> **See Also: SelExL User Guide** for more details on the item list file and SelExL's other configuration files.

[1] Extract Key – The column(s) on the Driving Table that determine which rows are extracted. This is normally the business item users refer to when they say "Please can I have all the data for <business term> 34567, 98654 and 45690." For example in a healthcare administration system it's likely to be patient id.

[2] Driving Table – Primary table which sits at the centre of the extraction process, normally the main table for the Extract Key. For example in a healthcare administration system it's likely to be the patients table.

File Extract and Load (BEX and BLO)



The **Validation, Load User Data** and **Populate Temporary Table** elements of BEX operate in exactly the same manner as when used in EXLO.

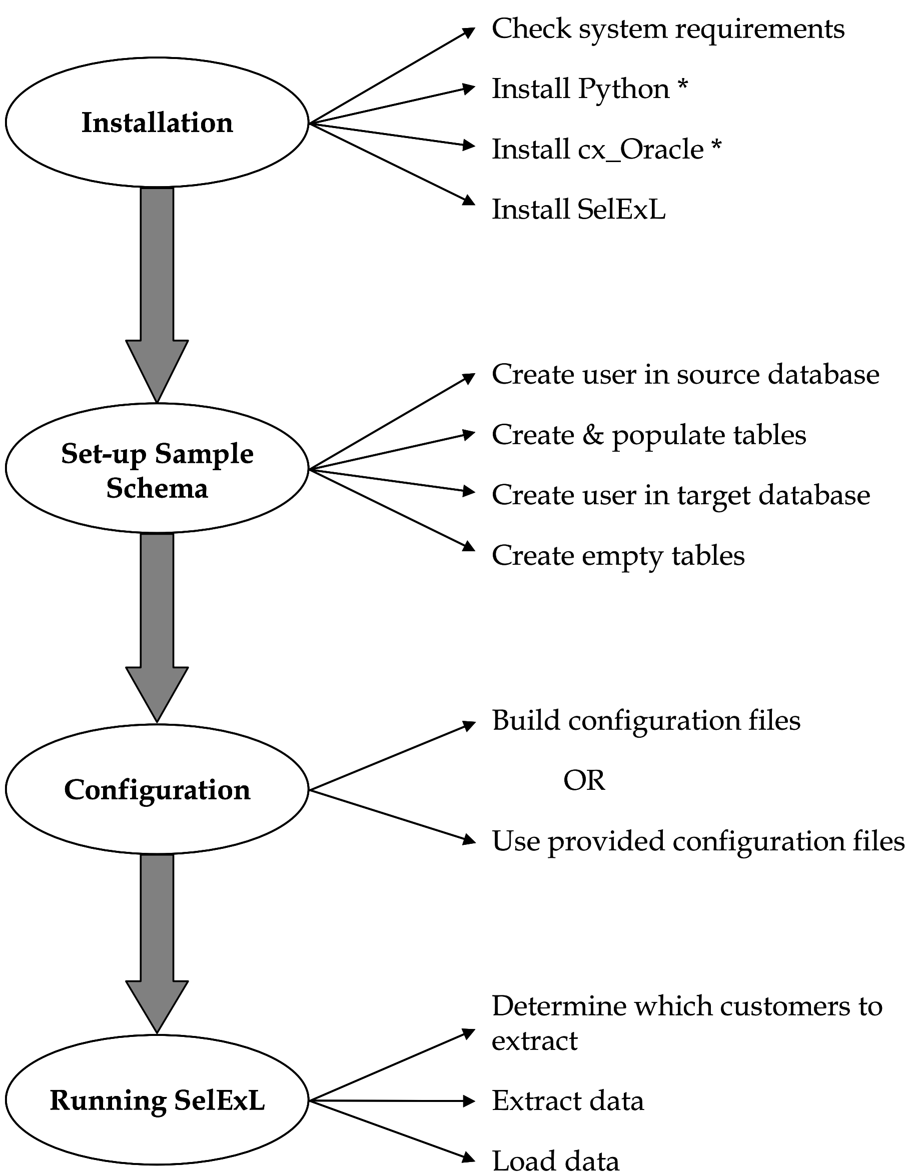
Extract Data to File - SelExL extracts the required rows for each table to a file using the keys from EXTRACT_KEYS. This is done using a streamed approach. The number of streams is user defined.

Load Data from File - SelExL loads the data into the target subset database from the files produced by the previous step. This can take place anytime after the extract to file has completed. Again it is carried out using a streamed approach, normally using SQL*Loader direct path.

1.3 Using the sample schema with SelExL

SelExL's sample schema provides an easy way to see SelExL in action without the need to understand all the workings of every aspect of SelExL. Scripts are provided to create the necessary Oracle user and all the database tables, indexes and alike. Suitable SelExL configuration files are also provided so you can be up and running as quickly as possible.

Using SelExL against the sample schema – Process Overview


Installation
Check system requirements
Install Python *
Install cx_Oracle *
Install SelExL
Set-up Sample Schema
Create user in source database
Create & populate tables
Create user in target database
Create empty tables
Configuration
Build configuration files
OR
Use provided configuration files
Running SelExL
Determine which customers to extract
Extract data
Load data

* Open source software required by SelExL. Installed automatically when installing the GUI version

See Also: Section 2.3, **"Installation Pre-requisites"** for more details on Python and cx_Oracle

C H A P T E R 2

Installation

This chapter includes the following information:

2 INSTALLATION

2.1 System Requirements

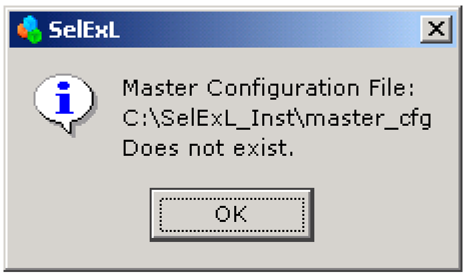
Before installing SelExL and building the sample schema you need to check the following:

	Requirement	Yes / No
Operating System	UNIX, Linux or Windows NT, 2000, XP	
Oracle software	Oracle client installation, including SQL*Loader. (Version 8.1.7, 9.2, 10.1 or 10.2) (9.0.1 is supported for use with SelExL but not recommended, as it is no longer supported by Oracle).	
Database access *	Oracle Net access to at least one Oracle database. (Version 8.1.7 or greater).	

* This is required for the sample schema. If a database is not available you will need to create one of your own. (Suggestion: use Oracle's database creation assistant, DBCA).

2.2 Installing the GUI Version

To install the Windows GUI version of SelExL double-click the supplied SelExL setup exe file and follow the simple installation instructions. The GUI installation includes all the code for the command line version. (Do not install into a directory whose name contains spaces e.g. *C:\Program Files*. See README.txt for full details.) To run the GUI version select SelExL from the usual Windows Start/Programs menu. If there is no default master configuration file (*master_cfg*) in the working directory then SelExL will display a warning message.



Master Configuration File:
C:\SelExL_Inst\master_cfg
Does not exist.

If you are not installing the command line version you may skip the remainder of this chapter.

2.3 Installation Pre-requisites

SelExL requires two additional pieces of software to be installed before it can be used, Python and cx_Oracle. You must install Python before installing cx_Oracle. These pre-requisites are NOT required to install the Windows GUI version of SelExL.

Python

Python is an Open Source scripting language similar to the likes of Perl and Tcl. SelExL requires version 2.3 or greater. To install Python, if you do not already have it installed, you first need to download it. You can do this from http://www.python.org/download. SelExL requires version 2.3 or greater, in general you should install the latest production version.

If the Oracle software you are using is 64 bit then you must ensure that Python is 64 bit i.e. it is a 64 bit binary package or the compiler you use when installing from source is 64 bit. (Note: If your platform is 32 bit then the Oracle software will be 32 bit but if your platform is 64 bit then the Oracle software maybe either 32 or 64 bit.)

The easiest way to install Python depends upon your platform. Here are some platform specific guidelines:

Platform	Installation Guidelines	Oracle 32/64 bit
Solaris	Either use a download from http://sunfreeware.com/ and then use pkgadd or follow the recommendations for "Other UNIX".	May be either 32 or 64 bit.
Other UNIX	Download the compressed tarball source for the latest production version. Follow the simple installation instructions at http://www.python.org/<latest_version>. You will need a C compiler to complete the installation.	Oracle 9i and above on HP-UX, AIX and Tru64 are 64 bit.
Linux	Python is usually provided with a Linux distribution so you probably do not need to install it from http://www.python.org/. If you do need to install then either follow the guidelines for "Other UNIX" or use an RPM package from http://www.python.org/<latest_version>/rpms.html.	May be either 32 or 64 bit.
Windows NT, 2000, XP, 2003	Download and install the Python binaries via http://www.python.org/<latest_version> or use ActivePython from ActiveState at http://www.activestate.com/Products/ActivePython. Both of these options provide an easy, user-friendly installation process.	Normally 32 bit unless you are using a 64 bit version of Windows.

cx_Oracle

cx_Oracle is Open Source database driver software that allows Python programs to access Oracle databases. SelExL requires version 4.1 or greater.

To obtain cx_Oracle go to http://www.computronix.com/utilities.shtml. If you are running on a Windows or Linux platform then download the appropriate Windows binary or Linux RPM, if available, and install. Otherwise you need to download the source.

To install from source carry out the following:

1. Ensure that the appropriate Oracle software is in your PATH. (Use *oraenv* on UNIX/Linux and Oracle Home Selector on Windows).
2. Un-pack the *cx_Oracle-<ver>.tar.gz* file to a *<cx_Oracle_dir>* using a package such as WinZip or 7-Zip on Windows and `tar xvfz cx_Oracle-<ver>.tar.gz` on UNIX/Linux.
3. Enter: `cd <cx_Oracle_dir>`
4. For Windows: `python setup.py build --compile=mingw32 install`
5. For UNIX/Linux: `python setup.py build install`
6. If necessary copy the resulting *cx_Oracle.pyd* to *<Python_install_dir>/lib/site-packages*.

2.4 Installing the Command Line Version

When you have completed the installation pre-requisites you are ready to install SelExL. To finish the installation process complete the following steps:

1. Create your own SelExL installation directory *<inst_dir>*. Do not install into a directory whose name contains spaces e.g. *C:\Program Files*. See README.txt for full details
2. `cd <inst_dir>`.
3. Un-pack *selexl_<ver>.tgz* to *<inst_dir>* using `tar xvfz selexl_<ver>.tgz` on UNIX/Linux and a package such as WinZip or 7-Zip on Windows.
4. Modify environment variables.

The SelExL software is now installed in *<inst_dir>*. The files for the SelExL demonstration based on the sample schema are located in *<inst_dir>/demo*.

Modify Environment Variables

Before you can run the command line version of SelExL you need to ensure that two environment variables are correctly set:

- PATH must include the directory where SelExL is installed.
- PYTHONPATH must also include the directory where SelExL is installed.

To see if these environment variables are set correctly on a Windows machine start-up a Dos command window and enter the following:

```
C:\> set PATH
C:\> set PYTHONPATH
```

This will display the values of PATH and PYTHONPATH respectively. If either variable is not set correctly then enter the following:

```
C:\> set PATH=%PATH%;c:\<inst_dir>
C:\> set PYTHONPATH=%PYTHONPATH%;c:\<inst_dir>
```

If PYTHONPATH was not set at all then enter:

```
C:\> set PYTHONPATH=c:\<inst_dir>
```

On UNIX and Linux this “checking and setting” of the environment variables is similar. The exact syntax will depend on which Shell you are using.

Additionally on UNIX/Linux you need to make sure that LD_LIBRARY_PATH includes *$ORACLE_HOME/lib* e.g. for the Korn shell

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib
```

Finally to check that SelExL has installed successfully enter:

```
C:\> sxl_exlo
```
or
```
$ sxl_exlo
```

This should display help information for *sxl_exlo*. If it does not then check that all the environment variables are set correctly as described above. Also try running:

```
C:\> python sxl_exlo
```
or
```
$ python sxl_exlo
```

CHAPTER 3

Setting up the Sample Schema

This chapter includes the following information:

- Create Sample Schema in the Source Database
- Create Sample Schema in the Target Database

3 SETTING UP THE SAMPLE SCHEMA

All the scripts required to set-up SelExL's sample schema are SQL*Plus scripts located in *<inst_dir>/demo/sql*. (*<inst_dir>* is the directory where you installed SelExL). To run SQL*Plus on Windows open a command window and enter:

```
C:\> cd <inst_dir>\demo\sql
C:\> sqlplusw
```

To run SQL*Plus on UNIX/Linux enter the following at the command line:

```
$ cd <inst_dir>/demo/sql
$ sqlplus <user>/@<database>
```

3.1 Create Sample Schema in the Source Database

To create the sample schema in the source database complete the following steps:

- Connect to the source database as a user with DBA privileges using SQL*Plus.
- Create the user to hold the sample schema:

```
SQL> @create_demo_user.sql
```

- Connect to the source database as the user you have just created, create the sample schema tables and populate them:

```
SQL> connect <user>@<database>
SQL> @create_demo_schema_windows.sql or
SQL> @create_demo_schema_unix.sql
```

3.2 Create Sample Schema in the Target Database

To create the sample schema in the target database (this can be the same as the source database) complete the following steps:

- Connect to the target database using SQL*Plus as described above. (You need to have CREATE USER and GRANT ANY ROLE privileges e.g. DBA)
- Create the user to hold the sample schema:

```
SQL> @create_demo_user.sql
```

- Connect to the target database as the user you have just created, create the sample schema tables and leave them empty:

```
SQL> connect <user>@<database>
SQL> @create_empty_demo_schema.sql
```

This completes the setup of the sample schema.

CHAPTER 4

Configuration

This chapter includes the following information:

- Configuration Overview
- Configuration Process

4 CONFIGURATION

4.1 Configuration Overview

Before you can run SelExL against the sample schema, or any other schema for that matter, you need to tell SelExL about the tables you wish to populate in the subset database. This covers the relationships between the tables and the columns that should be used to drive the extract for each table. SelExL provides three core configuration files for this purpose. A brief description of each is given below:

extractdriver_cfg

This file drives the whole extract process and normally comprises a single line. It contains:

- Name of the file holding the values of the Extract Key, known as an item list file.
- Name of the Driving Table.
- Name of the Extract Key used to drive the extract. This is a column on the Driving Table.
- Name of the column on EXTRACT_KEYS where SelExL inserts the values for the Extract Key.

For example:

```
c:\data\customers_to_extract CUSTOMERS CUSTOMER_ID NUM1
```

tablekeys_cfg

This file contains a list of all the tables that could be extracted by SelExL, the order is not important. The file contains:

- Name of a table that could be extracted by SelExL.
- Name of the column on this table that drives the extract for the table, normally the table's primary key. This column name must exist either as an entry in the third field of *extractdriver_cfg* or the second field of *populationkeys_cfg*. If it's a table from which all rows are required, for example a table holding reference data, then use the special entry ALL.

For example:

```
CUSTOMERS CUSTOMER_ID
ORDERS ORDER_ID
DECODES ALL
```

populationkeys_cfg

This file contains the information to enable the complete population of EXTRACT_KEYS. It is also used to help generate the final extract SQL. The order of entries is important. The file contains:

- Name of a source table that provides the values to insert into EXTRACT_KEYS for a particular required column.
- Name of the column on this source table.
- Normally the name of a foreign key column on the source table whose “parent table” already appears as an **earlier** entry in this file or in *extractdriver_cfg*.
- Name of the column on EXTRACT_KEYS that holds the values that will be joined to the source table. Usually this is the same as the column on EXTRACT_KEYS where the values from the source table will be inserted.

For example:

```
ORDERS       ORDER_ID    CUSTOMER_ID  NUM1
ORDER_ITEMS  PRODUCT_ID  ORDER_ID     NUM1
```

See Also: SelExL User Guide for more details on these configuration files, and SelExL configuration as a whole.

4.2 Configuration Process

The main aim of the configuration process is to ensure that each of the three core configuration files contains appropriate entries so that EXTRACT_KEYS is populated with the correct key column values. This will then ensure that all the required rows are extracted for each table when SelExL is run.

SelExL comes with a set of configuration files for the sample schema. You can choose to either work through the remainder of this chapter and produce your own configuration files or move on, to Chapter 5, **“RUNNING SelExL”**, and use the configuration files provided in *<inst_dir>/demo/cfg.*

To create the configuration files for SelExL’s sample schema use the following framework.

Create extractdriver_cfg entry

Determine which table (Driving Table) and column (Extract Key) is the “centre” of the sample schema i.e. the column the business users refer to as the main item in the database and the main table to which it belongs. Use this to create the *extractdriver_cfg* entry. Make sure the column you chose on EXTRACT_KEYS has the same data type as Extract Key. The file you use for the item list file need not exist for the time being.

In the case of the sample schema we are dealing with a simple customer ordering system where customer is the central business item. CUSTOMERS and CUSTOMER_ID are the Driving Table and Extract Key respectively.

Produce an ordered list of tables

Produce a list of all the tables you wish to extract data from. This list needs ordering such that each table has at least one of its parent tables before it in the list, and preferably all of them. The first table should be the Driving Table. It does not actually matter where the stand-alone tables reside in the list; a good idea is to group them all together at the bottom.

Traverse the ER diagram to help produce this ordered list. This is all about helping to simplify the next task. For the sample schema ignore DEPARTMENTS, EMPLOYEES and JOBS. These tables are provided to illustrate more advanced features that are covered in the SelExL User Guide.

Create entries for tablekeys_cfg and populationkeys_cfg

For each <table> in your ordered table list carry out the following:

Step 1 - Identify the column(s) you wish to drive the extract for this <table>. Add an entry to *tablekeys_cfg* comprising the <table> and column(s) you have just identified.

Step 2 - Check to see if the column(s) you have just identified exist in either *extractdriver_cfg* or is an entry in the second field of *populationkeys_cfg*. If it does then go back to step 1 and pick the next table from the list otherwise continue with step 3.

Step 3 - Add a new entry to *populationkeys_cfg*.

- Field 1, the source table, normally a parent of <table> or a previously configured table with the appropriate relationship to <table>. This source table must contain the column identified in step 2. Occasionally it is <table>.
- Field 2 is the column identified in step 2
- Field 3 is a column from the source table, field 1. This column needs to either exist in *extractdriver_cfg* or be an entry in the second field of *populationkeys_cfg*.
- Field 4 is the column on EXTRACT_KEYS where the values for the column specified in field 3 will reside. Usually this is the same as the column on EXTRACT_KEYS where the values from the source table will be inserted.
- Optional fields 5 and 6 are not normally used *
- Now go back to step 1 and pick the next table from the list.

* In the case of WAREHOUSES_STOCK in the sample schema, which has a RAW data type for WAREHOUSE_ID, fields 5 and 6 are used. PRODUCT_ID and RAW1 are the values for fields 5 and 6 respectively.

The configuration files are now ready for review.

See Also:

- **SelExL User Guide** for full details on the use of *populationkeys_cfg's* optional fields.
- Appendix A, **" Step-by-Step Configuration"** for a more detailed approach to the configuration work including a complete worked example for the sample schema.

Configuration Review

Manually check *tablekeys_cfg* and *populationkeys_cfg* to ensure that all the essential table-to-table relationships have been captured. It is likely that some table-to-table relationships are not explicitly represented. It is quite possible that this presents no problems whatsoever as these "missing" relationships are implicitly represented. The next step should identify whether or not this is the case.

Configuration Validation and Testing

Before validating your configuration work you need to make sure that the table EXTRACT_KEYS has been created in the source database. Also ensure that there is at least one entry in the item list file referenced in *extractdriver_cfg,* this must be a valid value for Extract Key e.g. 123 when CUSTOMER_ID is the Extract Key.

To validate the configuration work, run SelExL in validation mode (`sxl_exlo -v`) and correct any errors – SelExL provides appropriate error messages.

Finally to test the configuration work, run SelExL to just populate EXTRACT_KEYS (`sxl_exlo -p`) using a small number of known values for Extract Key. Check that EXTRACT_KEYS contains all the expected values for each of the required keys e.g. CUSTOMER_ID, ACCOUNT_ID, ORDER_ID etc. This will help identify any genuine missing relationships.

See Also: Chapter 5, **"RUNNING SelExL"** for details on *sxl_exlo's* various options including validation.

C H A P T E R 5

Running SelExL

This chapter includes the following information:

- Introduction
- Running SelExL for the first time
- Run EXLO – Combined Extract and Load

5 RUNNING SelExL

5.1 Introduction

SelExL operates as a standard GUI program on the Windows platform and from the command line i.e. a standard telnet or X-Windows session in Linux and UNIX and in a Dos command window in Windows. Essentially the command line version of SelExL operates in exactly the same way no matter the platform. The GUI and command line versions provide identical functionality.

When running SelExL commands there are two ways to invoke SelExL:

```
$ python <SelExL module> <command line switches>
$ <SelExL module> <command line switches>
```

Where <SelExL module> is the name of a SelExL module e.g. *sxl_exlo* and <command line switches> are command line options for the module in question e.g. –h. For example:

```
$ python sxl_exlo -h
$ sxl_exlo -h
```

Both of these methods are equivalent and should always work. However the first method is the more robust.

If you are using the Windows GUI version then simply start SelExL via the Windows Start/Programs menu.

In this Quick Start Guide we only look at *sxl_exlo*, the single step process.

See Also:

- **SelExL User Guide** for details on SelExL's two-step file based option.
- Section 2.4, **"Installing the Command Line Version"** for details about setting the required environment variables.

5.2 Running SelExL for the First Time

First Steps

Before you can run SelExL to populate a subset database using the sample schema you must carry out the following simple steps:

Step 1 - Check that all the configuration work has been done and that you know where the configuration files *extract_driver_cfg, tablekeys_cfg* and *populationkeys_cfg* are.

Step 2 - Create a master configuration file – see **"Creating the Master Configuration File"**

Step 3 - Create a configuration file *tablelist_cfg*. This is simply a list of the tables you wish to populate in the subset database. An easy way to do this is take a copy of *tablekeys_cfg* and remove everything except the table names that populate the first field of each line. There is one provided for the sample schema in *<inst_dir>/demo/cfg*.

Step 4 - Create your item list file to hold the list of values that will determine the data extracted – see **"Creating an Item List File"**

Creating the Master Configuration File

The master configuration file (*master_cfg*) provides SelExL with required runtime information. For example the directory name where the configuration files reside. The name of the master configuration file can be any valid file name but must not includes spaces, * or $ in the name.

To run SelExL against the samples schema take a copy of the provided file, *master_cfg*, and modify it for your situation i.e. change directory names, database names and users.

Full details on the master configuration file can be found in the SelExL User Guide.

Creating an Item List File

An item list file contains the list of values of the business item that you are using to drive the extract. This business item, known as the "Extract Key", is the column name listed in *extractdriver_cfg* i.e. CUSTOMER_ID for the sample schema.

Suppose you wish to extract two customers with CUSTOMER_IDs 144 and 145 then your item list file would contain:

```
144
145
```

For further details and how to make use of SelExL's wildcard feature see the SelExL User Guide.

Running SelExL

You are now in a position to run your extract against the sample schema and populate the subset database using SelExL. For the purpose of this example we recommend you use *sxl_exlo*.

5.3 Run EXLO – Combined Extract and Load

sxl_exlo is the SelExL program to run a combined extract and load. Use *sxl_exlo* for the following to:

- Perform a “validation only” run of SelExL to check that all the configuration files are syntactically correct etc.
- Only populate the temporary table EXTRACT_KEYS i.e. no actual extract is performed.
- Perform an extract from the source database and load the extracted data into the target database. This is *sxl_exlo’s* normal use.
- Perform an extract and load using the rows already present in EXTRACT_KEYS.

The command line usage for *sxl_exlo* is:

Arguments:

-h	Provide this help
-l	Display software licence details
-m	Optional - Name of the master configuration file. Default is *master_cfg* in the current directory.
-s	Mandatory (when run in batch) - Password to connect to the source database. User is <Source_db_user> from the master configuration file Database is <Source_db_name> from the master configuration file
-t	Mandatory (when run in batch) - Password to connect to the target database. User is <Target_db_user> from the master configuration file Database is <Target_db_name> from the master configuration file
-a	Optional - Rows are appended to tables in the target database i.e. target tables are not truncated
-v	Optional - Configuration file validation only.
-e	Optional - Do not populate Extract_Keys. Perform extract and insert using the rows already in Extract_Keys.
-p	Optional - Only populate Extract_Keys and do not perform the extract.

At most 1 of the last 3 arguments can be specified at any one time.

To run SelExL against the sample schema enter:

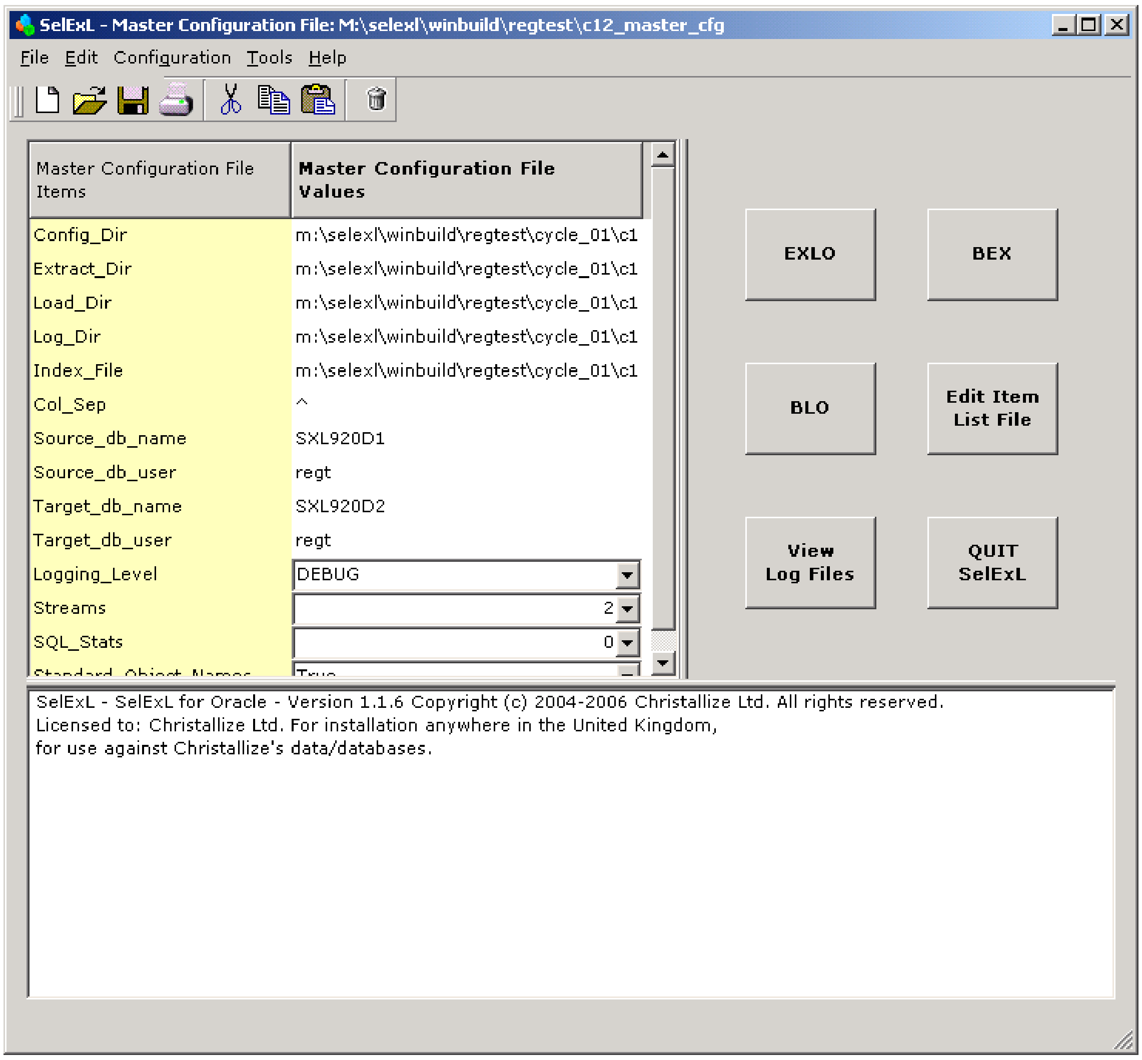
```
sxl_exlo –m <master configuration file>
```

You will then be prompted for the required database passwords. The process should complete within one or two minutes depending on your hardware and how busy the machines are.

You can go to the target database and see the data that has been loaded into your subset database.

Alternatively you can run EXLO from the Windows GUI version. See screenshots below:

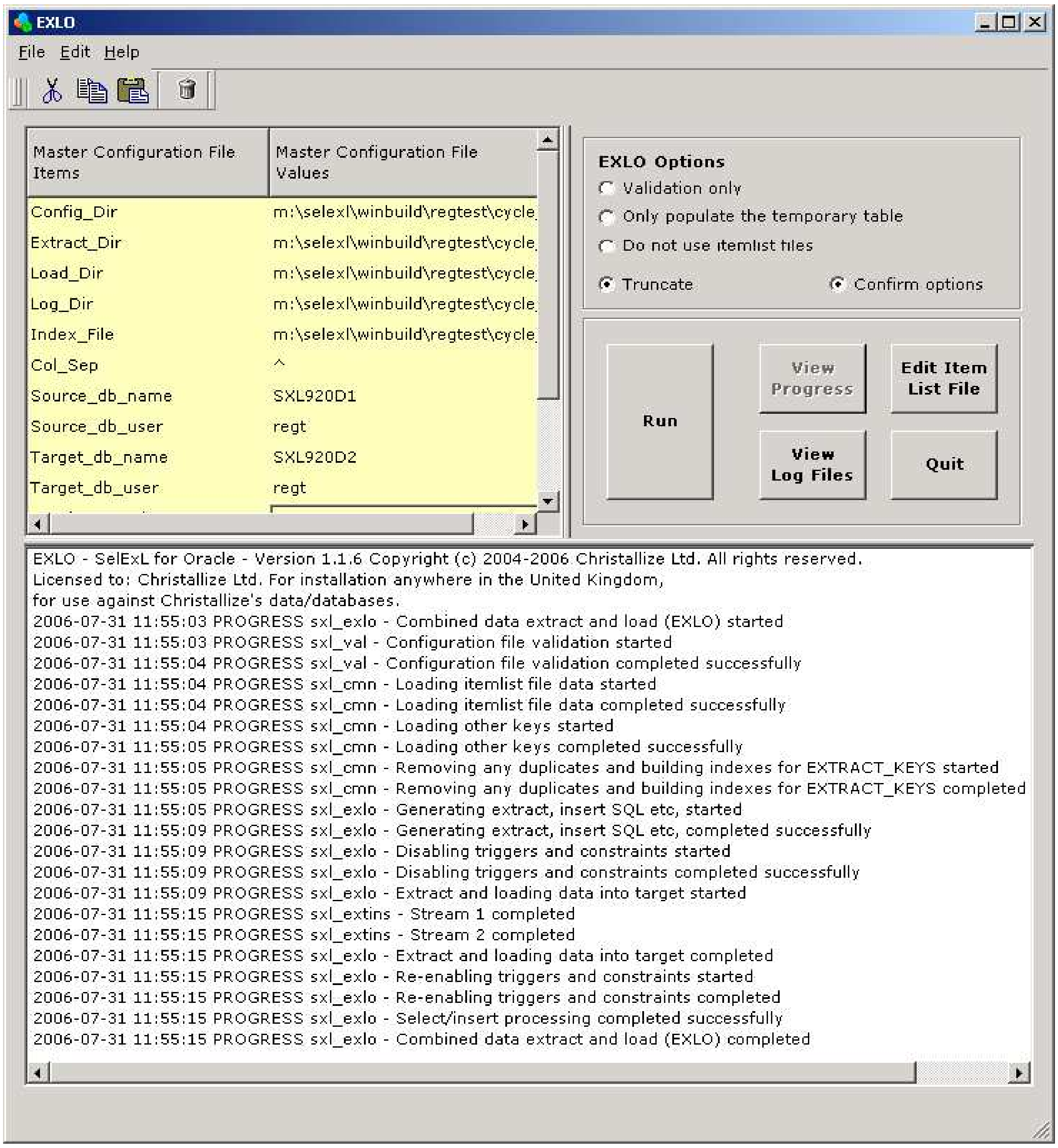
SelExL main window


SelExL - Master Configuration File: M:\selexl\winbuild\regtest\c12_master_cfg
File Edit Configuration Tools Help
EXLO
BEX
BLO
Edit Item List File
View Log Files
QUIT SelExL

Master Configuration File Items	Master Configuration File Values
Config_Dir	m:\selexl\winbuild\regtest\cycle_01\c1
Extract_Dir	m:\selexl\winbuild\regtest\cycle_01\c1
Load_Dir	m:\selexl\winbuild\regtest\cycle_01\c1
Log_Dir	m:\selexl\winbuild\regtest\cycle_01\c1
Index_File	m:\selexl\winbuild\regtest\cycle_01\c1
Col_Sep	^
Source_db_name	SXL920D1
Source_db_user	regt
Target_db_name	SXL920D2
Target_db_user	regt
Logging_Level	DEBUG
Streams	2
SQL_Stats	0

```
SelExL - SelExL for Oracle - Version 1.1.6 Copyright (c) 2004-2006 Christallize Ltd. All rights reserved.
Licensed to: Christallize Ltd. For installation anywhere in the United Kingdom,
for use against Christallize's data/databases.
```

EXLO main window



Master Configuration File Items	Master Configuration File Values
Config_Dir	m:\selexl\winbuild\regtest\cycle
Extract_Dir	m:\selexl\winbuild\regtest\cycle
Load_Dir	m:\selexl\winbuild\regtest\cycle
Log_Dir	m:\selexl\winbuild\regtest\cycle
Index_File	m:\selexl\winbuild\regtest\cycle
Col_Sep	^
Source_db_name	SXL920D1
Source_db_user	regt
Target_db_name	SXL920D2
Target_db_user	regt

EXLO Options
- ( ) Validation only
- ( ) Only populate the temporary table
- ( ) Do not use itemlist files
- [✓ Truncate] [✓ Confirm options]

```
EXLO - SelExL for Oracle - Version 1.1.6 Copyright (c) 2004-2006 Christallize Ltd. All rights reserved.
Licensed to: Christallize Ltd. For installation anywhere in the United Kingdom,
for use against Christallize's data/databases.
2006-07-31 11:55:03 PROGRESS sxl_exlo - Combined data extract and load (EXLO) started
2006-07-31 11:55:03 PROGRESS sxl_val - Configuration file validation started
2006-07-31 11:55:04 PROGRESS sxl_val - Configuration file validation completed successfully
2006-07-31 11:55:04 PROGRESS sxl_cmn - Loading itemlist file data started
2006-07-31 11:55:04 PROGRESS sxl_cmn - Loading itemlist file data completed successfully
2006-07-31 11:55:04 PROGRESS sxl_cmn - Loading other keys started
2006-07-31 11:55:05 PROGRESS sxl_cmn - Loading other keys completed successfully
2006-07-31 11:55:05 PROGRESS sxl_cmn - Removing any duplicates and building indexes for EXTRACT_KEYS started
2006-07-31 11:55:05 PROGRESS sxl_cmn - Removing any duplicates and building indexes for EXTRACT_KEYS completed
2006-07-31 11:55:05 PROGRESS sxl_exlo - Generating extract, insert SQL etc, started
2006-07-31 11:55:09 PROGRESS sxl_exlo - Generating extract, insert SQL etc, completed successfully
2006-07-31 11:55:09 PROGRESS sxl_exlo - Disabling triggers and constraints started
2006-07-31 11:55:09 PROGRESS sxl_exlo - Disabling triggers and constraints completed successfully
2006-07-31 11:55:09 PROGRESS sxl_exlo - Extract and loading data into target started
2006-07-31 11:55:15 PROGRESS sxl_extins - Stream 1 completed
2006-07-31 11:55:15 PROGRESS sxl_extins - Stream 2 completed
2006-07-31 11:55:15 PROGRESS sxl_exlo - Extract and loading data into target completed
2006-07-31 11:55:15 PROGRESS sxl_exlo - Re-enabling triggers and constraints started
2006-07-31 11:55:15 PROGRESS sxl_exlo - Re-enabling triggers and constraints completed
2006-07-31 11:55:15 PROGRESS sxl_exlo - Select/insert processing completed successfully
2006-07-31 11:55:15 PROGRESS sxl_exlo - Combined data extract and load (EXLO) completed
```

APPENDIX A

Step-by-Step Configuration

This appendix includes the following information:

- Step by step Configuration Process
- Worked Example

Appendix A Step-by-Step Configuration

This appendix contains a detailed step-by-step approach, including a worked example, for creating the three core configuration files *extractdriver_cfg*, *tablekeys_cfg* and *populationkeys_cfg* for SelExL's sample schema.

See Also: Chapter 4, **"CONFIGURATION"** for an overview of SelExL configuration.

To create the configuration files for your database use the following process:

Step 1 - Identify the Extract Key. This is the item for which you want to extract data, for a banking system this might be account#.

Step 2 - Identify the Driving Table for the Extract Key. This is the primary table for the Extract Key and is normally the table at the "centre" of a database's entity-relationship model. It is likely to have many "child tables" but no "parent tables". For an account based banking system this would probably be the accounts table.

Step 3 - Determine the column on EXTRACT_KEYS where the values for the Extract Key will reside. This column must have the same data type as the driving column on the Driving Table.

Step 4 - Create the entry for *extractdriver_cfg* based on the information derived from the previous three steps. The item list file can be any file you want that will hold Extract Key values.

Step 5 - Populate the item list file with a single entry that is a meaningful value for the Extract Key. This value does not need to actually be in the database for the purposes of this configuration process.

Step 6 - Add the first entry to *tablekeys_cfg*. This comprises the Driving Table and Extract Key.

Step 7 - Identify all the tables you need to extract data from. This is best done by working outwards from the Driving Table and completing each "branch" of the various parent-child hierarchies before starting on a new "branch". If you have some form of entity/table diagram showing the basic table relationships this will make this step much easier. If you do not possess any such diagram then this step may take a little longer.

Step 8 - Pick the next table from the list of tables produced by step 7. Identify the column you wish to drive the extract for this table. Add another entry to *tablekeys_cfg* comprising the table and column you have just identified.

Step 9 - Check to see if the column you identified in step 8 exists either in *extractdriver_cfg* or is an entry in the second field of *populationkeys_cfg*. If it does then go back to step 8 otherwise perform step 10.

Step 10 - Add a new entry to *populationkeys_cfg*. Normally this comprises the following steps:

Step 10a - Field 1, the source table, normally a parent of the table from step 8 or a previously configured table with the appropriate relationship to the table from step 8. The table must contain the column identified by step 8. Occasionally it is the table from step 8.

Step 10b - Field 2 is the column identified in step 8.

Step 10c - Field 3 is a column from the source table identified by step 10a. This column needs to either exist in *extractdriver_cfg* or be an entry in the second field of *populationkeys_cfg*.

Step 10d - Field 4 is the column on EXTRACT_KEYS where the values for the column identified in step 10c reside. Usually this is the same as the column on EXTRACT_KEYS where the values from the source table will be inserted.

Step 11 - Go back to step 8 until all the tables on the list have been processed.

Example

The following example is based on the sample database schema and table definition for EXTRACT_KEYS listed in Appendix B – Sample Schema. Tables JOBS, EMPLOYEES and DEPARTMENTS are excluded.

Step 1 - Identify the Extract Key.

We want to extract data for our test system based on customers. Therefore the driving column is CUSTOMER_ID.

Step 2 - Identify the Driving Table for the Extract Key.

The main table for customer information containing CUSTOMER_ID as the primary key is CUSTOMERS.

Step 3 - Determine the column on EXTRACT_KEYS where the values for the Extract Key will reside.

CUSTOMER_ID is a NUMBER data type so NUM1 is the required column on EXTRACT_KEYS.

Step 4 - Create the entry for *extractdriver_cfg*.

```
c:\data\customers_to_extract CUSTOMERS CUSTOMER_ID NUM1
```

Step 5 - Populate the item list file with a single entry.

File is *c:\data\customers_to_extract* with entry *144*

Step 6 - Add the first entry to *tablekeys_cfg*.

```
CUSTOMERS CUSTOMER_ID
```

Step 7 - Identify all the tables you need to extract data from.

First "branch" of tables is: CUSTOMERS, ACCOUNTS, BILLS, BILL_LINES.

Second "branch" is: ORDERS, ORDER_ITEMS, PRODUCT, WAREHOUSES_STOCK, WAREHOUSES with "sub-branches": SUPPLY_ORDER, STORES along with PRODUCT_DESC and STORES_STOCK

Remaining tables(s) is: DECODES

Step 8 - **Next table is ACCOUNTS**

Drive extract using ACCOUNT_ID, *tablekeys_cfg* entry is:

```
ACCOUNTS ACCOUNT_ID
```

Step 9 - Does ACCOUNT_ID exist in *extractdriver_cfg* or *populationkeys_cfg*?

- NO, step 10 is required.

Step 10a - *populationkeys_cfg* field 1.

In this case it is the table from step 8 i.e. ACCOUNTS.

Step 10b - *populationkeys_cfg* field 2.

Column from step 8 i.e. ACCOUNT_ID.

Step 10c - *populationkeys_cfg* field 3.

Column on ACCOUNTS that exists in *extractdriver_cfg* i.e. CUSTOMER_ID.

Step 10d - *populationkeys_cfg* field 4.

CUSTOMER_ID is a NUMBER data type so NUM1 is the required column on EXTRACT_KEYS. Hence *populationkeys_cfg* entry is:

```
ACCOUNTS ACCOUNT_ID CUSTOMER_ID NUM1
```

Step 8 - **Next table is BILLS**

Drive extract using ACCOUNT_ID, *tablekeys_cfg* entry is:

```
BILLS ACCOUNT_ID
```

Step 9 - Does ACCOUNT_ID exist in *extractdriver_cfg* or *populationkeys_cfg*?

- Yes, step 10 not required so back to step 8.

Step 8 - **Next table is BILL_LINES**

Drive extract using ACCOUNT_ID, *tablekeys_cfg* entry is:

```
BILL_LINES ACCOUNT_ID
```

Step 9 - Does ACCOUNT_ID exist in *extractdriver_cfg* or *populationkeys_cfg*?

- Yes, step 10 not required so back to step 8.

Step 8 - **Next table is ORDERS**

Drive extract using CUSTOMER_ID, *tablekeys_cfg* entry is:

```
ORDERS CUSTOMER_ID
```

Step 9 - Does CUSTOMER_ID exist in *extractdriver_cfg* or *populationkeys_cfg*?

- Yes, step 10 not required so back to step 8.

Step 8 - **Next table is ORDER_ITEMS**

Drive extract using ORDER_ID, *tablekeys_cfg* entry is:

```
ORDER_ITEMS ORDER_ID
```

Step 9 - Does ORDER_ID exist in *extractdriver_cfg* or *populationkeys_cfg*?

- NO, step 10 is required.

Step 10a - *populationkeys_cfg* field 1.

In this case it is the parent table i.e. ORDERS.

Step 10b - *populationkeys_cfg* field 2.

Column from step 8, ORDER_ID.

Step 10c - *populationkeys_cfg* field 3.

Column on ORDERS that exists in *extractdriver_cfg*, CUSTOMER_ID.

Step 10d - *populationkeys_cfg* field 4.

NUM1, as ORDER_ID is a NUMBER data type. Hence *populationkeys_cfg* entry is:

```
ORDERS ORDER_ID  CUSTOMER_ID  NUM1
```

Step 8 - **Next table is PRODUCT**

Drive extract using PRODUCT_ID, *tablekeys_cfg* entry is:

```
PRODUCT PRODUCT_ID
```

Step 9 - Does PRODUCT_ID exist in *extractdriver_cfg* or *populationkeys_cfg*?

- NO, step 10 is required.

Step 10 - Add a new entry to *populationkeys_cfg* following the process used for table ORDER_ITEMS.

```
ORDER_ITEMS PRODUCT_ID ORDER_ID NUM1
```

Step 8 - **Next table is WAREHOUSES_STOCK**

Drive extract using PRODUCT_ID, *tablekeys_cfg* entry is:

```
WAREHOUSES_STOCK PRODUCT_ID
```

Step 9 - Does PRODUCT_ID exist in *extractdriver_cfg* or *populationkeys_cfg*?

- Yes, step 10 not required so back to step 8.

Step 8 - **Next table is WAREHOUSES**

Drive extract using WAREHOUSE_ID, *tablekeys_cfg* entry is:

```
WAREHOUSES WAREHOUSE_ID
```

Step 9 - Does WAREHOUSE_ID exist in *extractdriver_cfg* or *populationkeys_cfg*?

- NO, step 10 is required including some additional configuration.

Step 10a - *populationkeys_cfg* field 1.

In this case it is the "junction table" between WAREHOUSES and PRODUCT i.e. WAREHOUSES_STOCK.

Step 10b - *populationkeys_cfg* field 2.

Column from step 8, WAREHOUSE_ID.

Step 10c - *populationkeys_cfg* field 3.

Column on WAREHOUSES_STOCK that exists in *populationkeys _cfg*, PRODUCT_ID.

Step 10d - *populationkeys_cfg* field 4.

NUM1, as PRODUCT_ID is a NUMBER data type.

Step 10e - *populationkeys_cfg* field 5. Required because PRODUCT_ID and WAREHOUSE_ID have different data types.

The column on WAREHOUSES_STOCK that will "join" to EXTRACT_KEYS. Usually the column identified by step 10c i.e. PRODUCT_ID

Step 10f - *populationkeys_cfg* field 6. Required because PRODUCT_ID and WAREHOUSE_ID have different data types.

The column on EXTRACT_KEYS holding values inserted from the source table. Therefore this column must have the same data type as WAREHOUSE_ID. In this case RAW1.

Hence *populationkeys_cfg* entry is:

```
WAREHOUSES_STOCK WAREHOUSE_ID PRODUCT_ID NUM1 PRODUCT_ID RAW1
```

Step 8 - **Next table is SUPPLY_ORDER**

Drive extract using ORDER_ID, *tablekeys_cfg* entry is:

```
SUPPLY_ORDER ORDER_ID
```

Step 9 - Does ORDER_ID exist in *extractdriver_cfg* or *populationkeys_cfg*?

- Yes, step 10 not required so back to step 8.

Step 8 - **Next table is STORES**

Drive extract using STORE_ID, *tablekeys_cfg* entry is:

```
STORES STORE_ID
```

Step 9 - Does STORE_ID exist in *extractdriver_cfg* or *populationkeys_cfg*?

- NO, step 10 is required.

Step 10 - Add a new entry to *populationkeys_cfg* following the process used for table ORDER_ITEMS.

```
SUPPLY_ORDER STORE_ID PRODUCT_ID NUM1
```

Step 8 - **Next table is PRODUCT_DESC**

Drive extract using PRODUCT_ID, *tablekeys_cfg* entry is:

```
PRODUCT _DESC PRODUCT_ID
```

Step 9 - Does PRODUCT_ID exist in *extractdriver_cfg* or *populationkeys_cfg*?

- Yes, step 10 not required so back to step 8.

Step 8 - **Next table is STORES_STOCK**

Drive extract using PRODUCT_ID, *tablekeys_cfg* entry is:

```
STORES_STOCK PRODUCT_ID
```

Step 9 - Does PRODUCT_ID exist in *extractdriver_cfg* or *populationkeys_cfg*?

- Yes, step 10 not required so back to step 8.

Step 8 - **Final table is DECODES**

We require all rows from DECODES so the *tablekeys_cfg* entry is:

```
DECODES ALL
```

This all results in the following configuration files:

- *extractdriver_cfg*

```
c:\data\customers_to_extract CUSTOMERS CUSTOMER_ID NUM1
```

- *populationkeys_cfg*

```
ACCOUNTS           ACCOUNT_ID     CUSTOMER_ID  NUM1
ORDERS             ORDER_ID       CUSTOMER_ID  NUM1
ORDER_ITEMS        PRODUCT_ID     ORDER_ID     NUM1
WAREHOUSES_STOCK   WAREHOUSE_ID   PRODUCT_ID   NUM1   PRODUCT_ID   RAW1
SUPPLY_ORDER       STORE_ID       PRODUCT_ID   NUM1
```

- *tablekeys_cfg*

```
CUSTOMERS         CUSTOMER_ID
ACCOUNTS          ACCOUNT_ID
BILLS             ACCOUNT_ID
BILL_LINES        ACCOUNT_ID
ORDERS            ORDER_ID
ORDER_ITEMS       ORDER_ID
PRODUCT           PRODUCT_ID
WAREHOUSES_STOCK  PRODUCT_ID
WAREHOUSES        WAREHOUSE_ID
SUPPLY_ORDER      ORDER_ID
STORES            STORE_ID
PRODUCT_DESC      PRODUCT_ID
STORES_STOCK      PRODUCT_ID
DECODES           ALL
```

Points to note

There are a number of situations when completing both *tablekeys_cfg* and *populationkeys_cfg* for a particular table that you have a choice of configurations. In the example above with ACCOUNTS we could have chosen CUSTOMER_ID as the driving column and used that in *tablekeys_cfg* in place of ACCOUNT_ID. Similarly when adding the *populationkeys_cfg* for table ORDER_ITEMS the chosen source table was ORDERS. In this case we could have used ACCOUNT_ID in place of CUSTOMER_ID as the value for field 3. In these situations it usually makes no difference which option you chose, in terms of the rows extracted by SelExL, providing the columns chosen do represent the relationships between the tables involved.

However the choices you make in configuration can a have an impact on performance. This is normally due to indexing. For example if ACCOUNTS did not for some reason have an index on CUSTOMER_ID then it would be wrong to use it as the driving column in the *tablekeys_cfg* entry for ACCOUNTS. Similarly if ORDERS was indexed on ACCOUNT_ID but not on CUSTOMER_ID then the *populationkeys_cfg* entry with ORDERS as the source table should be:

```
ORDERS ORDER_ID ACCOUNT_ID NUM1
```

See Also: SelExL User Guide for details on more unusual situations such as columns having the same meaning but different names on different tables and the case where multiple columns are required to drive an extract.

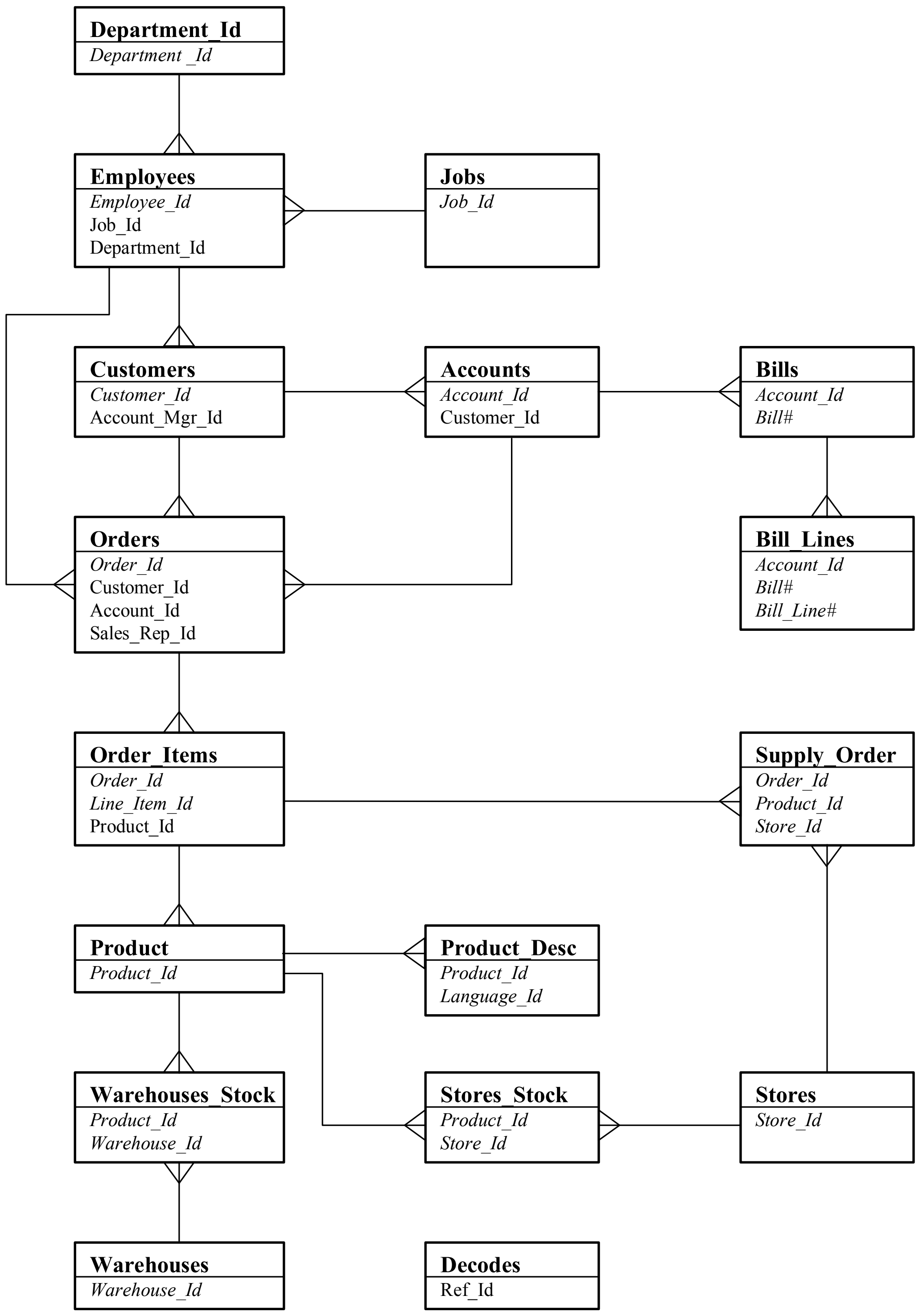
APPENDIX B

Sample Schema

This appendix includes the following information:

- Sample Schema Entity-Relationship diagram
- EXTRACT_KEYS definition

Appendix B Sample Schema



This is a slightly contrived schema designed for the purpose of illustrating the use of SelExL rather than being a correct database design. The “Entity-Relationship” diagram above only shows basic one-many relationships and makes no attempt to go any further by stating which relationships are mandatory etc. Only primary key, in *italics*, and foreign key columns are shown.

The relationship between EMPLOYEES and CUSTOMERS is between columns EMPLOYEES and ACCOUNT_MGR_ID. Similarly for EMPLOYEES and ORDERS the relationship is between columns EMPLOYEES and SALES_REP_ID.

The table definition of EXTRACT_KEYS to support the above schema is:

Column Name	Data Type
KEY_ID	VARCHAR2(50)
VCHAR1	VARCHAR2(50)
NUM1	NUMBER
RAW1	RAW(16)