User Guide

SelExL 1.2

User Guide

SelExL by Christallize

Manual Edition 1 – July 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

This user guide provides all the information on how to use SelExL covering both the initial configuration work and the on-going day-to-day use of SelExL.

> **See Also: SelExL Installation and Quick Start Guide** for details on installing SelExL.

This user guide consists of the following sections:

- **Chapter 1: Introduction and Overview** provides a brief overview of SelExL and the reason for subset databases.
- **Chapter 2: Planning and Initial Setup** covers the planning required in creating a subset database and setting up the SelExL user in the source database.
- **Chapter 3: Configuration** explains the process of building SelExL's required configuration files.
- **Chapter 4: Day-to-Day Use of SelExL** describes how to run SelExL.
- **Chapter 5: Advanced Configuration** explains the more advanced aspects of SelExL configuration.
- **Appendix A: Step-by-Step Configuration** provides a step-by-step guide on how to build SelExL's three core configuration files, including a worked example.
- **Appendix B: Configuration Files** provides a detailed description of SelExL's configuration files.
- **Appendix C: Sample Schema** describes the sample schema with use of a basic Entity-Relationship diagram.
- **Appendix D: Creating an Empty Database** describes how to create a small empty (no data) copy of an Oracle database.
- **Appendix E: Limitations and Restrictions** lists SelExL's technical restrictions and limitations.

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.

[1] Highlighted – Highlighted terminology explained in a footnote.

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

CHAPTER 1

Introduction and Overview

This chapter includes the following information:

- Why have Database Subsets?
- How SelExL works
- Creating subset databases with SelExL

1 INTRODUCTION AND OVERVIEW

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.

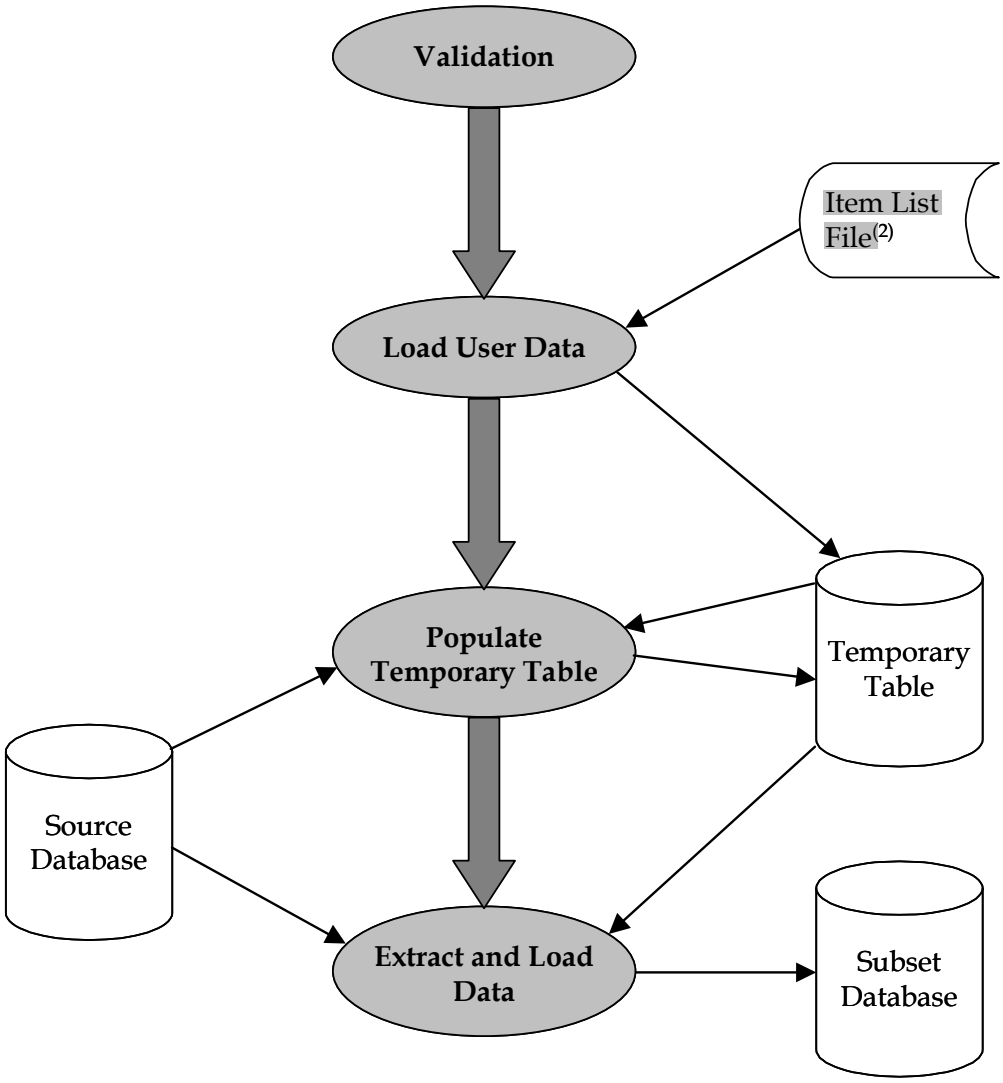
1.2 How SelExL works

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.

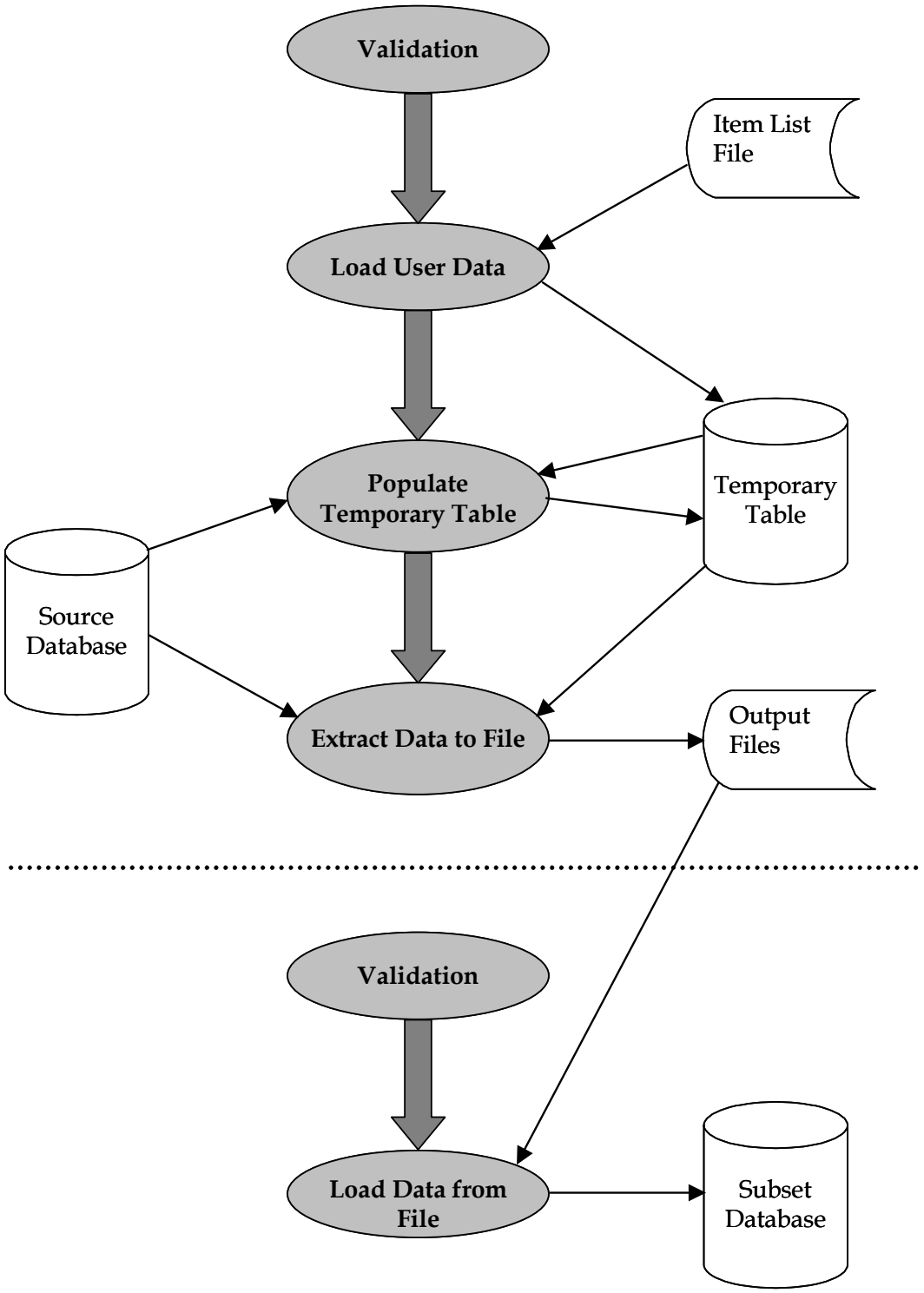
Extract and Load Data - SelExL extracts the required rows for each table using the keys from EXTRACT_KEYS and then inserts the data into the target subset database. This is carried out using multiple streams.

> **See Also:** Appendix B, **"Configuration Files"** for details about 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 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.

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.

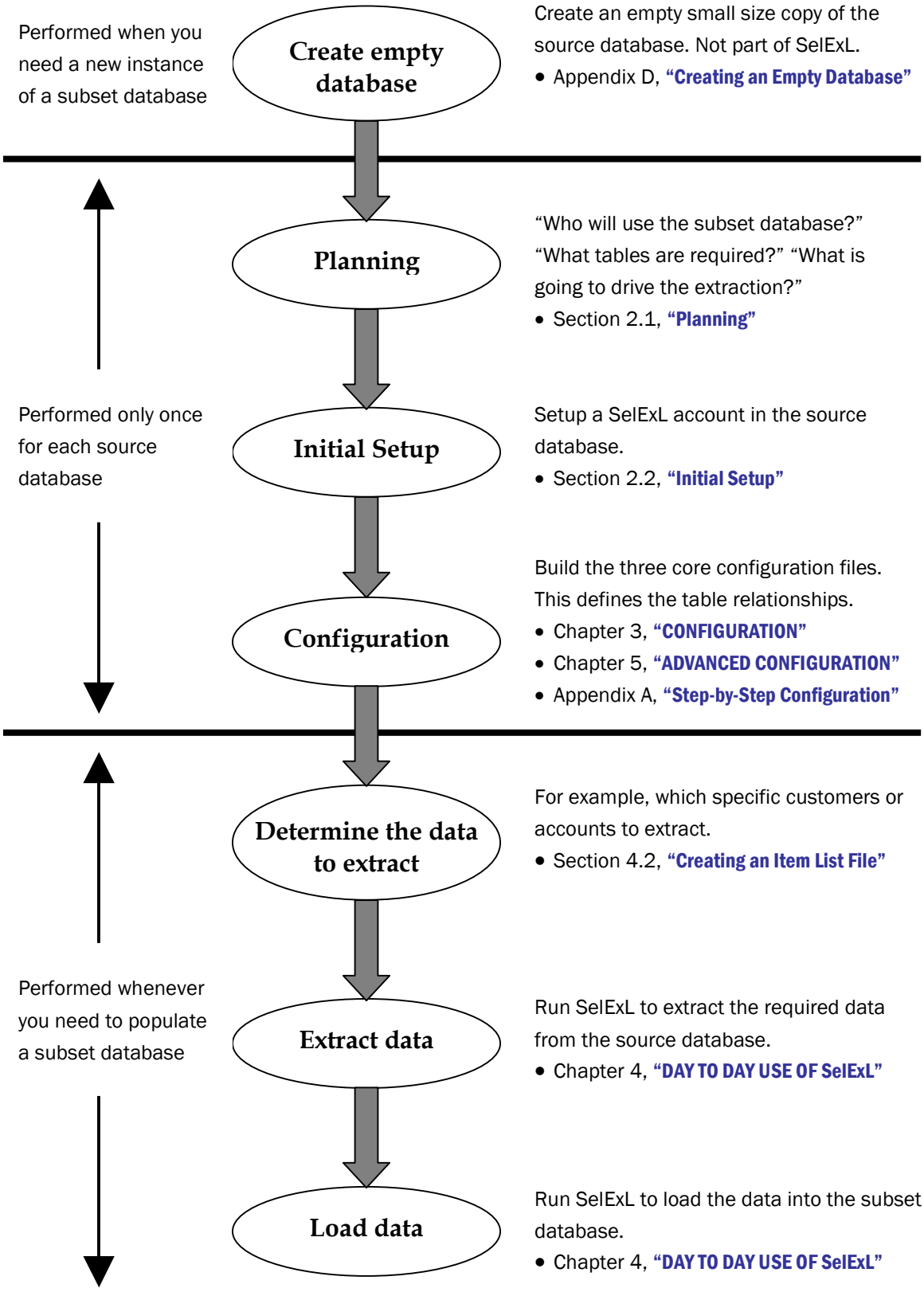
Load Data from File - SelExL loads the data into the target subset database from the files produced by the previous step. This can take place anytime after the extract to file has completed. Again it is carried out using a streamed approach, normally using SQL*Loader direct path.

1.3 Creating Subset Databases with SelExL

The process of creating and populating a subset database with SelExL comprises 7 stages. The first four stages are essentially one-off setup and configuration tasks covering the creation of the empty target database along with the building of SelExL's three core configuration files. It is only the final three stages that are required in the day-to-day use of SelExL. These final stages cover the **Sel**ection, **Ex**traction and **L**oading of the required data.

Chapters 2 and 3 of this User Guide cover the one-off setup and configuration tasks whilst chapter 4 deals with tasks involved in the day-to-day use of SelExL.

Creating a Subset Database – Process Overview



CHAPTER 2

Planning and Initial Setup

This chapter includes the following information:

- Planning the creation of subset database using SelExL
- Initial setup of the SelExL account in the source database

2 PLANNING AND INITIAL SETUP

The planning and configuration work requires knowledge of the database schema in question. Working with someone who knows the schema well will produce a better quality subset database and reduce the time taken to complete the planning and configuration work. The ideal candidate is someone who is fully conversant with the table-to-table relationships. This may well be a systems analyst or developer instead of the DBA.

Throughout this chapter we make reference to a sample schema to help illustrate particular points and provide worked examples.

> **See Also:** Appendix C, **"Sample Schema"** for details about SelExL's sample schema and a basic Entity-Relationship diagram.

2.1 Planning

When creating the first subset database for a particular production database you need the answers to a few key questions: "Who will be using the subset database?", "What will they be using it for?" and "What data do they need in the database?". The answers to these questions will drive the content of the subset database i.e. what are the required tables and also determine which business item is used to drive the extract, for example in a banking system is it customer or account?

We call this planning the Extract Definition(1) for the database schema in question. Planning is key. Extract Definitions are always created for a purpose and have a goal in mind.

Deciding the Extract Key and Driving Table

When you have answered these questions you should be able to identify the primary table, known as the Driving Table, along with the column(s) on the Driving Table that are used to determine what data is extracted. These column(s) are known as the Extract Key.

To determine a particular extraction the SelExL user provides a list of values, which are contained in an item list file, for the Extract Key of the Driving Table. These values determine the rows extracted from the Driving Table for the extract in question.

Rows for the remaining tables in the database are extracted if they are related to the extract rows from the Driving Table. It is these relationships, as they cascade down through the dependant tables, which provide the referential correctness for the subset database.

(1) Extract Definition – Defines the scope of the data that SelExL will extract to populate a subset database. It also defines the rules regarding the relationships between the various database tables included in the extract and the columns used to determine which rows are extracted for each table. Three core configuration files; *extractdriver_cfg, tablekeys_cfg* and *populationkeys_cfg* make up an Extract Definition.

The choice of Driving Table and Extract Key can significantly change the content of the subset database. It is important to make sure that this choice correctly reflects the requirements of the end-users (developers, testers, trainers etc.). It is common for the Driving Table to be the main central table in the database and the Extract Key to be the Driving Table's primary key. For example ACCOUNT_ID on the ACCOUNTS table. However it is not always the case that the main central table makes the best Driving Table.

An Extract Definition can consist of more than one Driving Table and associated Extract Key. SelExL is quite happy with this. However this does introduce additional complexity.

> **See Also:** Section 5.1, **"Multiple Driving Tables and Item List Files"** for details on how to use more than one Driving Table.

Identifying tables and table relationships

It is important to list all the tables that need populating in the subset database and understand the main relationships between these tables. There are a number of ways to go about this:

- Follow the foreign keys. The foreign keys, if present, automatically define relationships between tables. They provide a network of relationships between tables that you can traverse to help identify tables to include in the Extract Definition. You can either start at the top / centre of the hierarchy with the Driving Table and work "downwards and outwards" or start with tables at the bottom of the various hierarchies and work "upwards and inwards". Use the Oracle view ALL/USER_CONSTRAINTS.
- Entity-Relationship (ER) diagram. This diagram or some form of table diagram provides a hard copy "map" of the database schema and should help determine table relationships. Traverse the diagram starting from the Driving Table to help identify tables to include in the Extract Definition.
- Column names. Tables that contain "key" columns with the same name are likely to be related. (Usually columns that are included in primary and foreign keys).

Excluding tables

The first reaction is to assume that all tables in the production database need including in the Extract Definition to correctly populate the subset database. However this is often not the case. For example some tables are just "scratch" tables used by the application to hold temporary data and others are report output tables that hold the results of batch report runs. Clearly there is often no need, nor even any point, in extracting transient data of this nature.

Bear in mind the use of the subset database. If it's only going to be used for testing on-line components of the system then there's no need to include tables that are only used as part of the batch system.

"Related tables"

For each table chosen you need to decide if the table is a "related table" and you only require a subset of the rows i.e. it has some sort of relationship to the Driving Table, this maybe via a number of other "related tables".

If the table is a "related table", and it usually is, then you need to determine the column(s) on the table to drive the extract for the table. This will normally be either the table's primary key or the column(s) that comprise a foreign key to a parent table.

For example the table ORDERS in the **Sample Schema** has ORDER_ID as a primary key, CUSTOMER_ID as a foreign key to CUSTOMERS and ACCOUNT_ID as a foreign key to ACCOUNTS. All three of these columns are eligible as extract columns[1] for ORDERS. If CUSTOMERS is the Driving Table with CUSTOMER_ID as the Extract Key then CUSTOMER_ID is the logical choice as the extract column for ORDERS.

Stand-alone tables

Stand-alone tables are typically tables with no real relationship to any other table. Normally these are reference or configuration tables. They hold details such as tax rates, discount thresholds, batch-processing cycles etc. Usually you will require all rows from these tables. However some of these tables may hold technical information such as filenames to be used for batch output jobs and alike, in which case it's likely that you will want to exclude the table from the Extract Definition.

Planning Summary

After completing the planning stage you should have:

- Determined the Driving Table and associated Extract Key.
- Produced a list of tables to include in the Extract Definition.
- Determined those tables that need all rows extracting.
- Determined the extract columns for the "related tables".

The decisions reached at this stage are not final and it's quite likely that you will make a few changes as you go through the configuration work.

[1] Extract column – Column(s) on a table that are used by SelExL to "drive" the extract for the particular table. They are used as part of the join condition between the table and EXTRACT_KEYS.

2.2 Initial Setup

SelExL requires an Oracle database account on the source database from where data will be extracted. This account has the following requirements:

- Select access for all tables from which data will be extracted.
- Select access against V$MYSTAT (SYS.V_$MYSTAT) if you wish SelExL to have the capability to report a small number of Oracle database session statistics.
- ALTER SESSION privilege if you wish SelExL to have the capability to enable Oracle SQL and wait event tracing via the use of event 10046CREATE SESSION privilege for SelExL to connect to the database using this account.
- CREATE TABLE privilege to allow creation of table EXTRACT_KEYS. This privilege can be revoked when EXTRACT_KEYS has been created.
- CREATE INDEX privilege to allow creation of user defined indexes on EXTRACT_KEYS.
- Sufficient quota on the user's default tablespace to allow EXTRACT_KEYS to hold the required rows for the "intermediate keys" and any indexes. As a guide 100 MB should be plenty unless you are planning to extract large amounts of data (millions of rows) in which case the space requirements could be as high as 1 GB depending on the amount of data extracted and the complexity of the database in question.

  See Also: Appendix B, **"Configuration Files"** for a description of the master configuration file and settings to allow the reporting of Oracle statistics and wait event tracing.

Setup steps

Complete the following steps to setup the required database account on the source database:

Step 1 - Create the database account with the requirements listed above. An example script to create the account is *create_extract_user.sql* in SelExL's *sql* directory. The script needs to be run using an account with DBA privileges e.g. SYSTEM.

Step 2 - Connect to the database using the account created in Step 1.

Step 3 - Create table EXTRACT_KEYS. An example script is *create_extract_keys.sql* in SelExL's *sql* directory. The exact column definitions will depend on the database in question but we suggest you use the definition in *create_extract_keys.sql* to start with. If after completing the remaining configuration work you need to modify EXTRACT_KEYS then you can do so at that time either using an ALTER TABLE statement or by dropping and re-creating EXTRACT_KEYS.

CHAPTER 3

Configuration

This chapter includes the following information:

- Configuration overview
- Configuration process

3 CONFIGURATION

3.1 Configuration Overview

For SelExL to extract the correct rows from the source database it needs to understand the relationships between the various database tables and the item driving the extract, the Extract Key e.g. CUSTOMER_ID. This is accomplished with the use of 3 core configuration files. A brief description of each is given below.

> **See Also:** Appendix B, **"Configuration Files"** for a full description of each of SelExL's configuration files.

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.
- 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
```

However just because a table is listed in *tablekeys_cfg* does not guarantee that it will be extracted.

> **See Also:** Section 4.2, **"Running SelExL for the First Time"** for information on the *tablelist_cfg* configuration file.

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 the following four mandatory fields:

- 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
```

There are two other optional entries that maybe required. These are discussed later through the use of a worked example.

> **See Also:**
>
> - Appendix A, **"Step-by-Step Configuration"** for full details of the worked example.
> - Chapter 5, **"ADVANCED CONFIGURATION"** for situations when *populationkeys_cfg*'s optional additional fields are used.

Configuration file layout

All of SelExL's configuration files follow the same rules regarding comments, field separators etc. Here are the rules:

- Blank lines are ignored.
- Lines starting with #, or white space followed by a #, are treated as comments and ignored by SelExL.
- Lines containing # (except as above) are not treated as comments, i.e. in-line comments are not permitted.
- Fields are separated by white space i.e. any number and mixture of spaces and tabs.
- Environment variables e.g. $HOME can only be used as part of file or directory names.
- SelExL's configuration files are case sensitive, even on a Windows platform. You need to be consistent in how you refer to table and column names across configuration files. The simplest approach is to have everything in either uppercase or lowercase.

  See Also: Appendix B, **"Configuration Files"** for how to deal with the very rare situation where a table or column name includes an unusual character e.g. / or actually has some lowercase in it's name.

Tables need to be listed in the way that SelExL will reference them. Therefore if SelExL needs to use synonyms then use these synonyms in the configuration files. If, however, SelExL needs to use OWNER.TABLE_NAME then you must use this full reference method in the configuration files. Note that SelExL cannot make use of views anywhere in its configuration files.

3.2 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 in the Extract Definition.

If you have a large number of tables as part of your Extract Definition then it's advisable to split the list of tables into a number of smaller lists and complete this configuration process for one list at a time. Working with a small number of tables to start with will help you gain confidence and familiarity with the SelExL configuration process.

To create the configuration files for your database use the following framework (you need to have completed the planning stage before continuing).

Create extractdriver_cfg entry

Use the results from the **Planning** stage (Extract Key and Driving Table) 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.

Produce an ordered list of tables

The list of tables produced as part of the planning work needs ordering. The order should be 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 matter where the stand-alone tables reside in the list; a good idea is to group them all together at the bottom.

If traversing an ER diagram or following foreign keys produced the starting table list then the table list may already be in the correct order. This is all about helping to simplify the next task.

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 another 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 1. Occasionally it is <table>.
- Field 2 is the column identified in step 1
- 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.

See Also:

- Chapter 5, **"ADVANCED CONFIGURATION"** for situations when *populationkeys_cfg*'s optional fields are used.
- Appendix B, **"Configuration Files"** for a complete description of *populationkeys_cfg* and SelExL's other configuration files.

The configuration files are now ready for review.

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 many 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. (**Configuration Validation and Testing** should identify whether this is the case or not.) For example consider CUSTOMERS, ACCOUNTS and ORDERS from the sample schema. The relationships are:

1. Each customer must have one or more accounts and may have placed one or more orders.
2. Each account may have one or more orders placed against it and must belong to one and only one customer.
3. Each order belongs to one and only one customer and is placed against one and only one account.

A set of configuration files to support this is:

extractdriver_cfg

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

tablekeys_cfg

```
CUSTOMERS  CUSTOMER_ID
ACCOUNTS  CUSTOMER_ID
ORDERS  CUSTOMER_ID
```

populationkeys_cfg (empty)

This configuration shows a relationship between ORDERS and CUSTOMERS as well as ACCOUNTS and CUSTOMERS but there is nothing specific between ORDERS and ACCOUNTS. However this causes no issues for SelExL as the relationship is implicit because both ACCOUNTS and ORDERS have an explicit relationship with a common table, CUSTOMERS.

> **See Also:** Chapter 5, **"ADVANCED CONFIGURATION"** if you need to work through more complicated situations.

Configuration Validation and Testing

Before validating your configuration work make sure that the table EXTRACT_KEYS has been created in the source database. Also add a single entry to 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:** Section 4.3, **"EXLO – Combined Extract and Load"** for full details on *sxl_exlo*.

CHAPTER 4

Day-to-Day Use of SelExL

This chapter includes the following information:

- SelExL and environment variables
- Running SelExL for the first time
- EXLO – Combined extract and load
- BEX – Extracting data to files
- BLO – Loading data from files
- Extra information on running SelExL
- Using the Windows GUI Version

4 DAY TO DAY USE OF SelExL

4.1 SelExL and Environment Variables

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.

If you are running the command line version you need to ensure that two environment variables are correctly set. However if you are running the GUI version then you can ignore the remainder of this section.

- 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>
```

<*inst_dir*> is the directory where SelExL is installed.

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.

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.

4.2 Running SelExL for the First Time

First Steps

Before you run SelExL to populate a subset database carry out the following simple steps, illustrated using the command line version (you can also use the GUI version for steps 3-5):

Step 1 - Check that SelExL is installed. Enter:

```
C:\> sxl_exlo
```

This should display help information for *sxl_exlo*.

See Also: Section 4.1, **"SelExL and Environment Variables"** if required, to check environment variables.

Step 2 - 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 3 - Create a master configuration file - see **"Creating the Master Configuration File".**

Step 4 - 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.

Step 5 - 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 file name can be any valid file name but must not includes spaces, * or $ in the name. *<inst_dir>/demo/cfg* has a template *master_cfg* with all the key parameters listed. Use a copy of this file as the basis for your *master_cfg*. Many parameters can be set in the *master_cfg* to tailor how SelExL executes.

See Also:

- Appendix B, **"Configuration Files"** for a complete description of each parameter in *master_cfg* and a sample file.
- Section 4.6, **"Extra Information on Running SelExL"** for details on SelExL streaming and logging levels.

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 field 3 of *extractdriver_cfg* e.g. CUSTOMER_ID.

For example if the Extract Key is CUSTOMER_ID and you wish to extract five customers with CUSTOMER_IDs 23467, 48753, 48672, 90365 and 92303 then your item list file would contain:

```
23467
48753
48672
90365
92303
```

If you wish to extract a "random" sample then you can use wildcards. For example to extract a 5% sample your item list file might look like this:

```
%,%01
%,%02
%,%03
%,%04
%,%05
```

The % at the beginning tells SelExL that this entry is a "wildcard" entry and to use the LIKE operator when matching CUSTOMER_IDs from the item list file with those in the Driving Table. You can combine normal entries with wildcard entries in the same item list file.

If the values entered into the item list file contain spaces or commas then these specific values need enclosing in single quotes. For example if ADDRESS was the Extract Key then an item list file might look like this:

```
'25, First Avenue, London'
'16, Bristol Road, Leeds'
%,'%, Swift Way, Brighton'
```

Remember the name of your item list file must be the same as the one listed in *extractdriver_cfg*.

> **See Also:** Appendix B, **"Configuration Files"** for a complete description of item list files including the case when the Extract Key comprises more than one column.

Running SelExL

You are now in a position to run your extract and populate the subset database using SelExL. The final decision is whether to extract the data to files and load the subset database from these files or combine the extract and load together and run it as a single process.

As a general rule use the combined extract and load approach (EXLO) for small sized extracts typically less than a few thousand non-wildcard entries in the item list file. Use the extract to files (BEX) followed by a subsequent load (BLO) method for larger extracts or when having the extract output to standard operating system files is beneficial. Both of these methods can be carried out using either the command line or GUI version of SelExL.

See Also:

- Section 4.3, **"EXLO – Combined Extract and Load"** for details on running *sxl_exlo*.
- Section 4.4, **"BEX – Extracting Data to Files"** for details on running *sxl_bex*.
- Section 4.5, **"BLO – Loading Data from Files"** for details on running *sxl_blo.*

Note: These sections provide details for the command line version. If you are using the GUI version you may benefit from reading the first part of each section.

4.3 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.

Examples:

```
python sxl_exlo -m /home/oracle/mastercfg -s srcpass -t tgtpass -a
```
(Note: Target tables have rows inserted, existing rows remain intact.)

```
python sxl_exlo -s
```
(Note: *sxl_exlo* prompts for the required database passwords. Target tables are truncated before rows are inserted into them. The default.)

```
python sxl_exlo -m /home/oracle/mastercfg -s srcpass -t tgtpass -v
```
(Note: Processing stops after the configuration files are validated.)

Note: On most platforms it is unnecessary to include *python* in the command line i.e.

```
sxl_exlo -s
```
will normally suffice.

4.4 BEX – Extracting Data to Files

sxl_bex is the SelExL program used to extract the required data to a set of files, one file per table. Use *sxl_bex* for the following to:

- Perform an extract from the source database to a set of files. This is *sxl_bex's* normal use.
- Perform an extract to file using the rows already present in EXTRACT_KEYS.

To save disk space *sxl_bex* can use "on the fly" gzip compression. The compression factor is typically between three and seven times depending on the data.

The command line usage for *sxl_bex* 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
-c Optional - Output files are compressed via gzip
-e Optional - Do not populate Extract_Keys. Perform extract using the rows already in Extract_Keys.

Examples:

```
python sxl_bex –m /home/oracle/mastercfg –s srcpass
```

python sxl_bex -m /home/oracle/mastercfg -s
(Note: s*xl_bex* prompts for the required database password.)

```
python sxl_bex –m /home/oracle/mastercfg –s srcpass –c
```
(Note: Output files are compressed.)

Note: On most platforms it is unnecessary to include *python* in the command line i.e.

```
sxl_bex –s
```
will normally suffice.

4.5 BLO – Loading Data from Files

s*xl_blo* is the SelExL program used to load a set of files produced by s*xl_bex* into a subset database. Use s*xl_blo* for the following to:

- Load a set of files produced by s*xl_bex* into a target database.

BLO only loads those tables that have a file in the load directory, as defined in the master configuration file. It takes no notice of the tables listed in *tablelist_cfg*.

When loading a set of gzip compressed files produced by BEX, BLO (running on UNIX or Linux) will uncompress the files "on the fly" so there's no need to reserve any disk space for the uncompressed files. However uncompressing "on the fly" is not used when running on a Windows platform (Windows does not support named pipes). You will need to ensure there is sufficient disk space to hold temporary copies of the uncompressed files while the file is loaded into the target table. BLO deletes the uncompressed file immediately the file has finished loading. The uncompressed files are stored in the same directory as the compressed files.

The command line usage for s*xl_blo* 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.
-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 - Append data to tables i.e. do not truncate before the load
-e Optional - Maximum number of errors to allow per table before aborting the load (normally data related). This is the value for the SQL*Loader ERRORS command line parameter. The default is 500.

Examples:

```
python sxl_blo -m /home/oracle/mastercfg -t tgtpass
```

```
python sxl_blo -m /home/oracle/mastercfg -t
```
(Note: *sxl_blo* prompts for the required database password.)

```
python sxl_blo -m /home/oracle/mastercfg -t tgtpass -a
```
(Note: Data is appended to tables without truncation.)

Note: On most platforms it is unnecessary to include python in the command line i.e.

```
sxl_blo -t
```
will normally suffice.

4.6 Extra Information on Running SelExL

This section contains information on some specific aspects of running SelExL including: streaming, log files, running in batch and alike.

Streaming

SelExL runs all table extracts and loads as a multi-streamed process to improve performance. The number of streams used is defined in the master configuration file and ranges from 1 to 100. What's the best number of streams to use? Here are a few guidelines:

- If you are running SelExL in a dedicated slot then start by setting streams to a value between one and two times the number of CPUs on the database server e.g. if the database server has 8 CPUs then set the number of streams between 8 and 16. You can then increase or decrease the number of streams used in future, depending on how busy the server was during the run.
- If you are running SelExL on a single CPU client PC, against a database on a remote server, then start off by using no more than 2 or 3 streams depending on the power of the client PC.
- If you are only running a small extract or load then generally there's little advantage to be gained in having more than two streams.
- If you are running SelExL when other users are using the machines SelExL will access, then use half the number of streams you would use if you were in a dedicated environment.

To make the best use of streaming SelExL creates its own order in which to extract / load the tables. The aim is to process the tables that will take the longest time first and thus maximise the usage of each stream. When using BLO to load data from files the load order is calculated by using file size and assuming that the bigger the file the longer it will take to load.

When it comes to extracting data using EXLO and BEX, SelExL uses table size to determine the order in which to process the required tables. There are two ways in which measure table size:

- Number of rows on the table.
- Amount of disk space allocated to the table.

SelExL will use the former (number of rows) providing there are table statistics available for the vast majority of the tables in question on the source database. Therefore it is important to ensure that the table statistics are up to date. For an Oracle database these statistics are collected via the use of the DBMS_STATS package or the ANALYZE command.

If table statistics do not exist i.e. for an Oracle database the column NUM_ROWS on DBA/ALL/USER_TABLES is NULL then SelExL will use the amount of space allocated to the tables to measure table size, via the BYTES column on DBA_SEGMENTS.

However if the SelExL user does not have SELECT privilege on DBA_SEGMENTS then this is not possible. In this case SelExL is not able to obtain any sensible ordering for extracting the tables. This means SelExL extract performance will be sub-optimal, unless you are only using a single stream in which case it does not matter.

In summary you should ensure that either table statistics exist or that the SelExL user has SELECT privilege on DBA_SEGMENTS.

> **See Also:** Appendix B, **"Configuration Files"** for details of the master configuration file and how to set the number of streams.

Log Files

SelExL produces a number of different log files. These all reside in the log directory defined in the master configuration file, or in one of its sub-directories. There's a unique log file produced for each run of a SelExL program e.g. BEX. In addition, during the streamed processing, there's an additional stream specific log file for each stream. Old stream log files are deleted at the beginning of each SelExL run.

BLO also produces a log file for each table it loads. They reside in a dedicated sub-directory of the log directory. (*<log_dir>/table_load_logs*).

You can set different logging levels (amount of detail logged to the log files) with the use of a parameter in the master configuration file.

> **See Also:** Appendix B, **"Configuration Files"** for details of the various different logging levels within SelExL.

Running SelExL concurrently

There are a number of restrictions, some of which are just common sense, when it comes to running multiple copies of SelExL at the same time. These are:

- You can only run one extract program at a time against a source database unless you connect to the database using different users.
- You can only run one load program into a specific schema of a target database at one time.
- Each concurrent copy of SelExL must use a different master configuration file.
- Each concurrent copy of SelExL must have different log, load and extract directories.

Database Constraints and Triggers

When loading data, using either EXLO or BLO, SelExL manages constraints and table level triggers in the following ways:

- All enabled triggers on tables being loaded are disabled before loading starts and enabled when the load is completed.
- All enabled foreign key constraints that relate to any table that's being loaded are disabled before loading starts. They are then enabled when the load is complete.
- All other constraints are left unchanged.

If you ever need to manually enable any constraints or triggers that SelExL has disabled then you will find an appropriately named SQL file in the SelExL working directory. This is a sub-directory of the log directory. (*<log_dir>/workdir*).

Using "OPS$" accounts

SelExL supports connecting to Oracle databases using external authentication and "OPS$" accounts. All you need to do is enter a blank password when prompted.

If you wish to use external authentication when running BLO then the database identified by your default ORACLE_SID will become the target database. BLO will not use the target database specified in the master configuration file. This is due to connection restrictions when using SQL*Loader.

Indexes on EXTRACT_KEYS

SelExL does not require any indexes to be in place on EXTRACT_KEYS. However performance may be significantly improved with the use of appropriate indexes. You can specify your own indexes for EXTRACT_KEYS by entering the appropriate CREATE INDEX statements into the index file defined in the master configuration file.

It is not possible to provide specific guidance on what indexes to create as it depends on individual circumstances. For example, is it a large extract with many rows on EXTRACT_KEYS, or are there only a few distinct values in column KEY_ID etc.

SelExL does not embed any hints in the SQL it runs and generates statistics for EXTRACT_KEYS after it's been populated and any indexes built. This means that providing the Oracle optimiser is behaving correctly then Oracle should not use any sub-optimal execution plans just because of the existence of a particular index on EXTRACT_KEYS. However you need to bear in mind the time taken to create the indexes and trade this off against performance gains during the extract phase.

Running SelExL in Batch

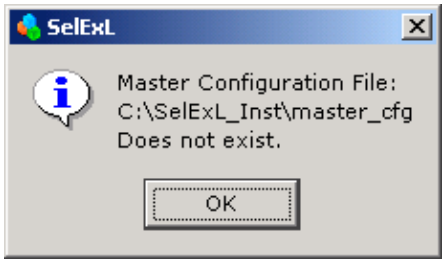
SelExL can be run in batch in just the same way as you would run any other command line program in batch. However on UNIX and Linux platforms we run up against the age-old problem of trying to hide passwords so that they cannot be displayed via the use of the *ps* command. The workaround is to make use of the "HERE" document feature of the UNIX shell environment and create a small shell script to run SelExL. For example a file called *run_exlo.sh* with the following contents:

```
sxl_exlo –m /home/fred/my_cfg  << HERE
source_password
target_password
HERE
```

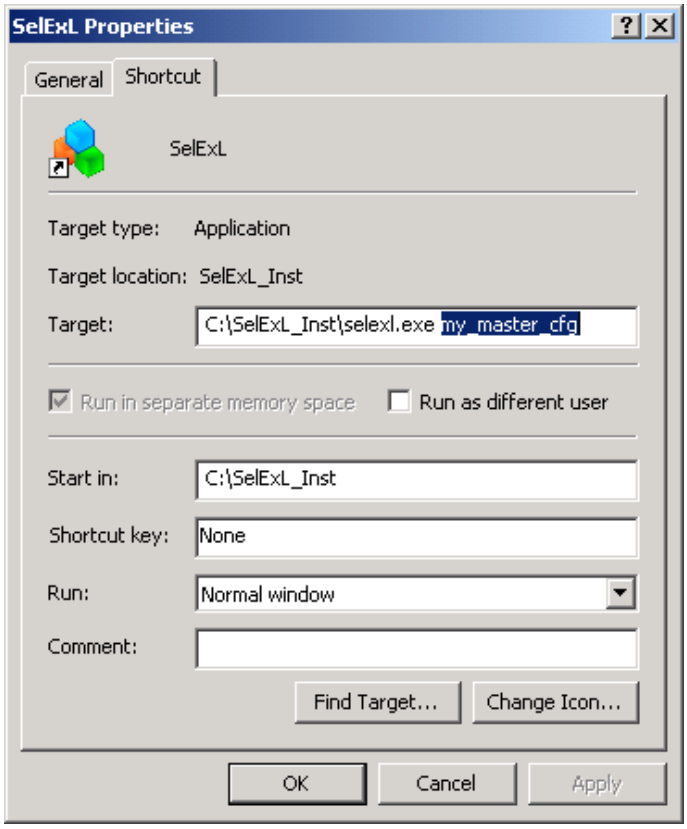
Note that each password only needs entering once and that the password for the source database is first.

4.7 Using the Windows GUI Version

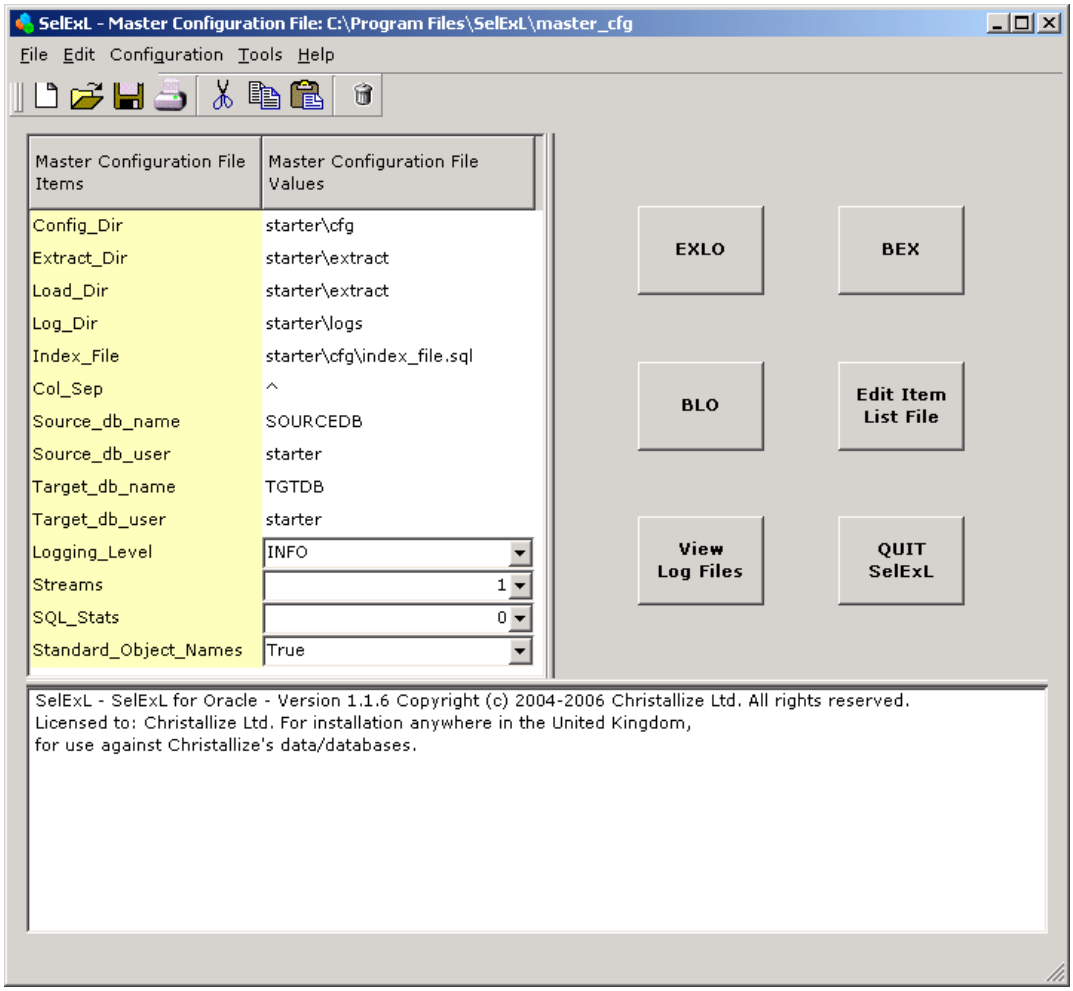
To start the GUI version of SelExL select SelExL from the usual Windows Start/Programs menu. If there is no default master configuration file (*master_cfg*) in the working directory then SelExL will display a warning message.



In this case either create a new master configuration file from within the GUI or open an existing master configuration file using the GUI. If you wish SelExL to start-up with a specific master configuration file then create a Windows shortcut for SelExL and amend its properties to include the name of the required master configuration file as a parameter for the Target. See below:



Here is a view of SelExL's main window:



You can access all of SelExL's functionality from this window. On-line help is provided for each of the main windows.

CHAPTER 5

Advanced Configuration

This chapter includes the following information:

- Multiple driving tables and item list files
- Multi column identifiers
- Columns with the same name but a different meaning
- Use of the optional fields in populationkeys_cfg
- Special features in tabkeys_cfg
- Adding extra entries to populationkeys_cfg

5 ADVANCED CONFIGURATION

This section looks at the more advanced areas of SelExL configuration not covered elsewhere.

See Also:

- Chapter 3, **"CONFIGURATION"** introduces the SelExL configuration process and SelExL's three core configuration files.
- Appendix A, **"Step-by-Step Configuration"** for a detailed step-by-step description of basic SelExL configuration.

5.1 Multiple Driving Tables and Item List Files

There are occasions when you require more than on item list file and/or Driving Table. These are:

A number of different item lists

You have a number of different item lists that contain their own specific values for the Extract Key and you wish to combine them in a single extract. Instead of combining item list files you can have multiple entries in *extractdriver_cfg* e.g.

```
c:\data\item_file1  CUSTOMERS  CUSTOMER_ID  NUM1
c:\data\item_file2  CUSTOMERS  CUSTOMER_ID  NUM1
c:\data\item_file3  CUSTOMERS  CUSTOMER_ID  NUM1
```

Multiple Extract Keys and Driving Tables

You wish for wider criteria to drive the whole extract, not just a single Extract Key. For example an insurance system, which is focused on both customers and individual insurance policies. Where a customer can have many policies (corporate customers have many policies) and a policy must belong to one and only one customer. In this case *extractdriver_cfg* might look like:

```
c:\data\customer_list  CUSTOMERS  CUSTOMER_ID  NUM1
c:\data\policy_list  POLICIES  POLICY_ID  NUM1
```

The impact of this approach depends on how you configure *populationkeys_cfg.* If you want the extract to include data for all the listed customers and insurance policies, including all policies belonging to customers not included in *c:\data\customer_list* but with a policy in *c:\data\policy_list*, then *populationkeys_cfg* will start with two entries like:

```
POLICIES   CUSTOMER_ID  POLICY_ID    NUM1
CUSTOMERS  POLICY_ID    CUSTOMER_ID  NUM1
```

If however you do not want to include all the additional policies for those customers not included in *c:\data\customer_list* but with a policy in *c:\data\policy_list*, then *populationkeys_cfg* will start with two entries like:

```
CUSTOMERS  POLICY_ID    CUSTOMER_ID  NUM1
POLICIES   CUSTOMER_ID  POLICY_ID    NUM1
```

5.2 Multi Column Identifiers

In a number of cases you will find tables that need referencing / accessing using a multi-column (composite) key. For example an ordering system where the main order table has region code and order number as the primary key. In these cases you need to specify these composite keys, and the associated columns on EXTRACT_KEYS, as a comma separated list with no spaces. Here's an example for *extractdriver_cfg*:

```
c:\data\orders  ORDERS  REGION_CODE,ORDER#  NUM1,NUM2
```

It is similar for all the other configuration files.

5.3 Columns with the same name but a different meaning

It is usual in database design for columns with the same name on different tables to have the same meaning. For example in the sample schema the column CUSTOMER_ID exists on the tables ACCOUNTS, CUSTOMERS and ORDERS. It has the same meaning on each of these tables i.e. a value of 1234 for CUSTOMER_ID on ACCOUNTS relates to and has the same meaning as for CUSTOMER_ID on CUSTOMERS.

However in some databases two or more tables may have columns with the same name but their meaning and usage may be completely different on each table. For example a database may by design have a column, ROW_ID (values generated by the application), on each table that is the unique identifier i.e. primary key for that table. If this were the case for the sample schema then a value of 6785 for ROW_ID on ACCOUNTS would have no relationship with a row on CUSTOMERS with the same value, 6785, for ROW_ID.

In this situation any columns with this characteristic need differentiating in the three core configuration files. This is accomplished by suffixing each entry of the column in question with <table>. For example entries in *tablekeys_cfg*:

```
ACCOUNTS  ROW_ID
CUSTOMERS ROW_ID
```

would need to become:

```
ACCOUNTS  ACCOUNTS.ROW_ID
CUSTOMERS CUSTOMERS.ROW_ID
```

Similarly for *extractdriver_cfg* and *populationkeys_cfg*.

5.4 Use of the optional fields in populationkeys_cfg

There are three main situations when it is necessary to use the two optional fields on *populationkeys_cfg*. These are described below:

"Join columns" have different data types

This normally occurs when you need to relate two tables where the primary keys for each table have different data types. For example in the sample schema the primary key for JOBS, JOB_ID, is VARCHAR2 whereas the primary key for EMPLOYEES, EMPLOYEE_ID is a NUMBER. A suitable entry for *populationkeys_cfg* to create a list of JOB_IDs is:

```
EMPLOYEES  JOB_ID  EMPLOYEE_ID  NUM1  EMPLOYEE_ID  VCHAR1
```

Number of columns in the "join columns" is unequal

This normally occurs when you need to relate two tables where the primary keys for each table have different number of columns. For example in the sample schema the primary key for ACCOUNTS is a single column whereas for BILLS the primary key has two columns. If both columns drive the extract for BILLS i.e. both columns are listed in the *tablekeys_cfg* entry then a suitable entry for *populationkeys_cfg* is:

```
BILLS ACCOUNT_ID,BILL#  ACCOUNT_ID  NUM1  ACCOUNT_ID  NUM1,NUM2
```

"Joining" columns with different names

This is not a normal occurrence and occurs when two tables are related but the columns that define the relationship have different names. For example in the sample schema there is a relationship between EMPLOYEES and ORDERS via the columns EMPLOYEE_ID and SALE_REP_ID respectively. To populate EXTRACT_KEYS with the required rows to enable an extract from EMPLOYEES we need the following entries in *populationkeys_cfg*.

```
ORDERS     SALES_REP_ID  CUSTOMER_ID   NUM1
EMPLOYEES  EMPLOYEE_ID   SALES_REP_ID  NUM1  EMPLOYEE_ID  NUM1
```

REFERENCES provides a different approach to this situation.

Recursive Relationships

Sometimes the relationships exist between two columns on the same table rather than two columns on different tables. This is a recursive or self-referencing relationship. For example employees manage other employees. This can be seen on the EMPLOYEES table in SelExL's sample schema with the columns EMPLOYEE_ID and MANAGER_ID.

To cater for this situation SelExL provides a SELFREF entry for *populationkeys_cfg*. This allows extra values to be inserted into EXTRACT_KEYS for a particular column based on a recursive relationship.

For example in the case of the EMPLOYEES table you may not only require those employees who are the account managers for customers you have selected in your item list file, but also all the employees that they manage. To populate EXTRACT_KEYS with the required rows to enable an extract from EMPLOYEES we need the following entries in *populationkeys_cfg*.

```
EMPLOYEES  EMPLOYEE_ID  ACCOUNT_MGR_ID  NUM1  EMPLOYEE_ID   NUM1
EMPLOYEES  EMPLOYEE_ID  MANAGER_ID      NUM1  SELFREF_DOWN
```

The key differences in this *populationkeys_cfg* entry are that MANAGER_ID is the column EMPLOYEE_ID is related to on the EMPLOYEES table i.e. MANAGER_ID "manages" EMPLOYEE_ID. SELFREF_DOWN tells SelExL to "walk down" the hierarchy i.e. populate EXTRACT_KEYS with the EMPLOYEE_IDs of all the employees "below" those EMPLOYEE_IDs already held in EXTRACT_KEYS. If you wished to "walk up" the hierarchy then you would enter SELFREP_UP.

The order of entries in *populationkeys_cfg* is important. If you do not want any other values being inserted into *populationkeys_cfg* based on the additional EMPLOYEE_IDs from this recursive relationship then place the SELFREF entry at the end of *populationkeys_cfg*.

> **See Also:** Appendix B, **"Configuration Files"** for a complete description of *populationkeys_cfg*.

5.5 Special features in tabkeys_cfg

There are three additional options that can be added to a *tablekeys_cfg* entry. These can help to refine the extract for the table in question. Note that these options only relate to the table in question and have no impact on the contents of EXTRACT_KEYS.

REFERENCES

Sometimes a child table is related to its parent table via columns that have different names on each table. For example in the sample schema, ORDERS and EMPLOYEES have a child/parent relationship via columns SALES_REP_ID and EMPLOYEE_ID. If the extract is being driven using EMPLOYEES as the driving table then one way of extracting data from ORDERS is via the SALES_REP_ID / EMPLOYEE_ID relationship. In this case the *tablekeys_cfg* entry is:

```
ORDERS SALES_REP_ID (REFERENCES EMPLOYEE_ID)
```

This means that instead of looking for values of SALES_REP_ID on EXTRACT_KEYS to extract data from ORDERS, SelExL will look for values of EMPLOYEE_ID instead. This feature can be used in place of adding extra entries to *populationkeys_cfg*. See **"Joining" columns with different names**.

AND / OR

This provides the ability to tie the extract of the table in question to two items on EXTRACT_KEYS and not just one. For example:

```
STORES_STOCK  PRODUCT_ID AND  STORE_ID
```

This means that when extracting rows for STORES_STOCK SelExL will match against both PRODUCT_ID and STORE_ID on EXTRACT_KEYS, thus potentially reducing the number of rows extracted when compared to the normal entry that excludes STORE_ID. Similarly OR can used in place of AND. This conversely has the potential to increase the number of rows extracted when compared to the normal entry that excludes STORE_ID. Note you can use REFERENCES entries in this situation e.g.

```
ORDERS SALES_REP_ID (REFERENCES EMPLOYEE_ID) OR CUSTOMER_ID
```

Table join

If you wish to place another condition on a table extract by joining it to another table then you can. This is accomplished by the use of the "=" option e.g.

```
PRODUCT  PRODUCT_ID  = SUPPLY_ORDER PRODUCT_ID
```

In this case rows are only extracted from PRODUCT if PRODUCT_ID matches the "PRODUCT_IDs" in EXTRACT_KEYS and also these PRODUCT_IDs must exist on SUPPLY_ORDER.

5.6 Adding extra entries to populationkeys_cfg

When you have completed the initial step-by-step configuration work you may find that for some reason the Extract Definition does not generate all the required rows for EXTRACT_KEYS. In this case you need to decide which additional entries you require in *populationkeys_cfg* to fill any gaps. The example given as part of **"Joining" columns with different names** is a special case of this.

SelExL provides a special "insert feature" for *populationkeys_cfg* that allows you to add your own INSERT statement to populate EXTRACT_KEYS with any extra values you require. This INSERT statement must adhere to two basic rules:

- Start with: `INSERT INTO extract_keys (key_id,`
- Include `SELECT … FROM` instead of a `VALUES` clause.

You should only use this special feature if you cannot accomplish what you require by using SelExL's standard configuration features.

> **See Also:** Appendix B, **"Configuration Files"** for a complete description of *populationkeys_cfg*.

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*.

See Also: Chapter 3, **"CONFIGURATION"** for more details on 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
```

APPENDIX B

Configuration Files

This appendix includes the following information:

- Configuration file summary
- Master configuration file
- Extract driver configuration file
- Item list file
- Table keys configuration file
- Population keys configuration file
- Table list file

Appendix B Configuration Files

Configuration File Summary

File	Purpose	When Modified
master_cfg	Holds all the standard runtime information such as database names and file and directory names.	After the configuration work is complete and before running SelExL for the first time. Sometimes before subsequent SelExL runs if you need to change the names of source and target databases etc.
extractdriver_cfg	Holds the high level details that drive the whole extract including the name of the item list file. Forms part of the Extract Definition.	During the configuration process and sometimes before a run of SelExL to change the name of the item list file.
item list file	Holds the list of items to extract e.g. account numbers.	Normally before each run of SelExL.
tablekeys_cfg	Defines the basis for extracting rows from each table. Forms part of the Extract Definition.	Normally only during the configuration process.
populationkeys_cfg	Defines the basic rules that govern the table-to-table relationships. Forms part of the Extract Definition.	Normally only during the configuration process.
tablelist_cfg	List the tables that are included in an extract.	After the configuration work is complete and before running SelExL for the first time. Sometimes before subsequent SelExL runs if you wish to omit certain table(s) from a particular extract.

Master Configuration File

The master configuration file provides SelExL with runtime information that SelExL requires to execute correctly. The parameters are:

Config_Dir * Name of the directory containing the core configuration files: *extractdriver_cfg, tablekeys_cfg, populationkeys_cfg* and *tablelist_cfg*.

Extract_Dir *	Name of the directory where BEX writes the extract files, one file per table.
Load_Dir *	Name of the directory holding the files for BLO to load.
Log_Dir *	Name of the directory to hold all SelExL's log files.
Index_File *	File containing CREATE INDEX statements for EXTRACT_KEYS. These statements are all user defined.
Col_Sep	Single character that's used as a column separator in the extract files. It must not exist in any character columns that SelExL extracts.
Source_db_name	Name of the source database from where SelExL extracts data. Normally a production database.
Target_db_name	Name of the subset database where data is loaded.
Source_db_user	Name of the Oracle account used to connect to the source database.
Target_db_user	Name of the Oracle account used to connect to the target database.
Logging_Level	Defines the level of logging SelExL writes to the various log files. There are: • CRITICAL – Only log critical application errors • ERROR – CRITCAL and ordinary errors • WARNING – ERROR and warning messages • PROGRESS – WARNING and progress messages • INFO – PROGRESS and information messages • DEBUG – INFO and debug information • FULL_DEBUG – DEBUG and the full level of debug information. This can output a large amount of data to the log files. Only use under guidance from SelExL support staff. Logging to the screen is preset to PROGRESS and cannot be changed.
SQL_Stats	Defines the level of SQL statistics gathered by SelExL and reported in SelExL log files, for each database session created by SelExL. There are: • 0 – No statistics gathered. • 1 – Basic session level statistics only (CPU, physical reads and logical reads). • 2 – Level 1 plus standard SQL trace. • 3 – Level 2 plus SQL trace wait information • 4 – Level 3 plus SQL trace bind information.

Streams	The number of concurrent streams used by SelExL when extracting or loading data. Maximum is 100.
Standard_Object_Names	This is an optional boolean parameter that tells SelExL how to deal with the case of table and column names in the various configuration files. The default of True means that all table and column names are "normal" i.e. exist as uppercase names in TABLE_NAME and COLUMN_NAME in Oracle's data dictionary and have no "funny" characters in the name e.g. /. In this situation table and column names can appear in the configuration files in any case providing the usage is consistent. False means that the name used in the configuration file must exactly match the name used by Oracle i.e. for "normal" tables this means uppercase. However for tables that have lowercase names in the database e.g. Accounts then it must appear as Accounts in the configuration files and not accounts or ACCOUNTS etc.

* These parameters can have values containing environment variables.

Here is a sample file:

```
Config_Dir              $HOME/cfgdir
Extract_Dir             $HOME/extract
Load_Dir                $HOME/load
Log_Dir                 $HOME/logs
Index_File              $HOME/selexl_indexes.sql
Col_Sep                 ^
Source_db_name          PRODDB
Target_db_name          DEVDB1
Source_db_user          data_owner
Target_db_user          data_owner
Logging_Level           INFO
SQL_Stats               1
Streams                 4
Standard_Object_Names   True
```

extractdriver_cfg

This file contains the name of the Driving Table(s) and associated Extract Key(s). The format is:

Field 1 <Item_File>	Name of the file containing the list of values for the specified Extract Key, which drives the extract. Referred to as the item list file.
Field 2 <Table>	Name of the Driving Table.
Field 3 <Key>	Comma separated list of the column(s) that drive the extract (Extract Key). This is normally a single column name.
Field 4 <Out_Col>	Comma separated list of the columns on EXTRACT_KEYS that will hold the key values for <Key>.

Normally there will be just one entry in the file. If there are multiple entries then the key population process will be driven by an OR process i.e. all the keys derived from the first Extract Key plus any additional values provided by any additional Extract Keys.

Item List File

This file is a list of values that will either be simply inserted into EXTRACT_KEYS or used to drive an INSERT INTO extract_keys ... SELECT ... statement. Each line contains a single value (or a comma separated list of values in the case where the Extract Key consists of more than a single column), which is either a specific value or a value containing wildcards. Values containing spaces or commas need single quotes around them e.g.

```
123456
234567
'Fred, Flintstone'
```

In the case of a wildcard e.g. %01, this equates to a LIKE statement being used in the where clause e.g.

```
WHERE cust_id LIKE %01.
```

Also wildcard entries are prefixed by %, e.g.

```
%,%01
23456
```

Below is an example item list file where ADDRESS1, ADDRESS2, ADDRESS3 is the Extract Key.

```
'24, John St', London, 'United Kingdom'
%, %, 'West Chiltington', 'United Kingdom'
'63, 1st Avenue', 'New York', USA
```

Notice that in line 1 United Kingdom is enclosed in quotes but London is not whereas on line 3 New York is enclosed in quotes but USA is not.

The second line means all addresses in West Chiltington, United Kingdom.

Here's a summary of the basic rules:

1. The column separator character must not exist in the file.
2. All lines must have the same format, allowing for the possibility of a leading % to denote a wildcard line.
3. Double quotes are not allowed anywhere.
4. The number of single quotes must be even.
5. Single quotes are only used as delimiters.
6. The number of fields must match the number of columns in <Key> in *extractdriver cfg*.
7. Wildcard entries start with an extra field containing a single %.

tablekeys_cfg

This file contains the key columns that drive the extract for each table. The format is:

Field 1 <Table>	Name of the table.
Field 2 <Key>	Comma separated list of the column(s) that drive the extract. This is normally a single column name. This name must match a name in field 2, <Tab_Key>, of *populationkeys_cfg* or field 3, <Key> in *extractdriver_cfg*. It must exist as a column on <Table>. Sometimes this field is augmented with a References entry: <Key> (REFERENCES <Ref_Key>) where <Ref_Key> is a comma separated list of column(s) that <Key> relates to. In this case <Key> only needs to exist as a column on <Table> and <Ref_Key> must match a name in field 2, <Tab_Key>, of *populationkeys_cfg* or field 3, <Key> in *extractdriver_cfg*. It is also possible to have two columns with either AND or OR in between. This allows greater control on the extract for the table by making it dependant on two columns not just one. The columns need to exist in *populationkeys_cfg* or *extractdriver_cfg* in the same way as for a normal single column name. These columns can also be augmented with a References entry if required. To specify that all the rows in the table require extracting enter ALL.
Field 3 <Equal >	Optional and not normally used. This is an equals sign stating that there is an additional constraint on the extract with a join to another table i.e. =
Field 4 <Match_Tab>	Optional and only used if <Equal> is present. Name of a table to which <Table> is joined to further restrict the extract.

Field 5 <Match_Key> Optional and only used if <Equal> is present. Comma separated list of columns to join. This list must have the same number of elements as in <Key>. (It will normally be an identical list.) The first element is joined with the first element in <Key> and the second with the second etc.

An example AND / OR entry including References is:

<Table> <Key1> (REFERENCES <Ref_Key1>) OR <Key2> (REFERENCES <Ref_Key2>)

populationkeys_cfg

This file contains the information to drive the population of EXTRACT_KEYS. The order of entries in this file is important. The standard format is:

Field 1 <Table> Name of the table that provides the values for the particular key(s). Sometimes called the source table.

Field 2 < Tab_Key > Comma separated list of the column(s) that are extracted from <Table> to populate EXTRACT_KEYS. Normally a single column.

Field 3 <In_Key> Name of the values in EXTRACT_KEYS.KEY_ID to drive the extract of key values from <Table>. This is a comma-separated list of column name(s), normally a single column, which will be inserted as a single value into EXTRACT_KEYS.KEY_ID.

The entry must exist as either a <Tab_Key> value in an earlier line in the file or as <Key> from *extractdriver_cfg*. It must exist as a column on <Table> unless there is an <Out_Key> value for this entry.

Field 4 < In_Col > Name of the column(s) on EXTRACT_KEYS where the values of <In_Key> are to be found. There must be same number of columns as in <In_Key>.

Field 5 <Out_Key> Optional and not normally required. Comma separated list of the columns on <Table> that will join to EXTRACT_KEYS. This list must have the same number of columns as in <In_Key>. If left blank then the value is assumed to be the same as <In_Key>.

Field 6 < Out_Col > Exists if and only if <Out_Key> exists. Comma separated list of the columns on EXTRACT_KEYS that will hold the values extracted from <Table>.<Tab_key>. This list must have the same number of elements as in <Tab_Key>. If left blank then the value is assumed to be the same as <In_Col>.

It's possible (but unlikely) to have entries with identical <Table>,<Tab_Key> pairs if <In_Key> is different. Similarly it's acceptable for two entries to have the same <Tab_Key> but with a different <Table>.

There are two non-standard format entries. The first is a "SELFREF" entry that is used for recursive or self-referencing relationships and the second is a user-defined INSERT statement.

The format of a SELFREF entry is: (Having exactly five fields identifies this type of entry).

Field 1 <Table>	Name of the table that provides the values for the particular key(s). Sometimes called the source table.
Field 2 < Tab_Key >	Name of the column that is extracted from <Table> to populate EXTRACT_KEYS. The entry must exist as either a <Tab_Key> value in an earlier line in the file or as <Key> from *extractdriver_cfg*. It must exist as a column on <Table>.
Field 3 <In_Key>	Name of the column on <Table> that is related to <Tab_Key>.
Field 4 <In_Col >	Name of the column on EXTRACT_KEYS where the values of <Tab_Key> are to be found. Also the column where the values will be stored.
Field 5 <SELFREF>	Either SELFREF_UP or SELFREF_DOWN. Indicates whether to "walk up" or "walk down" the hierarchy.

See Also: Section 5.4 **"Use of the optional fields in populationkeys_cfg"** for an example of SelExL configuration of a recursive relationship.

A user-defined INSERT statement is of the form:

```
INSERT INTO extract_keys (key_id, <col_list>)
SELECT DISTINCT '<col_name>', <col_list> FROM …….
```

<col_name> and <col_list> are user defined. *DISTINCT* is optional.

tablelist_cfg

This file contains the list of tables to extract. It is simply one table per line. The table must exist in *tablekeys_cfg*. It can be edited before each extract to change which tables are required for the particular extract.

Note: References to columns in all these configuration files can be prefixed with <Table>. if required e.g. ORDERS.ORDER_ID.

See Also: Section 5.3 **"Columns with the same name but a different meaning"** for an example of using these prefixes.

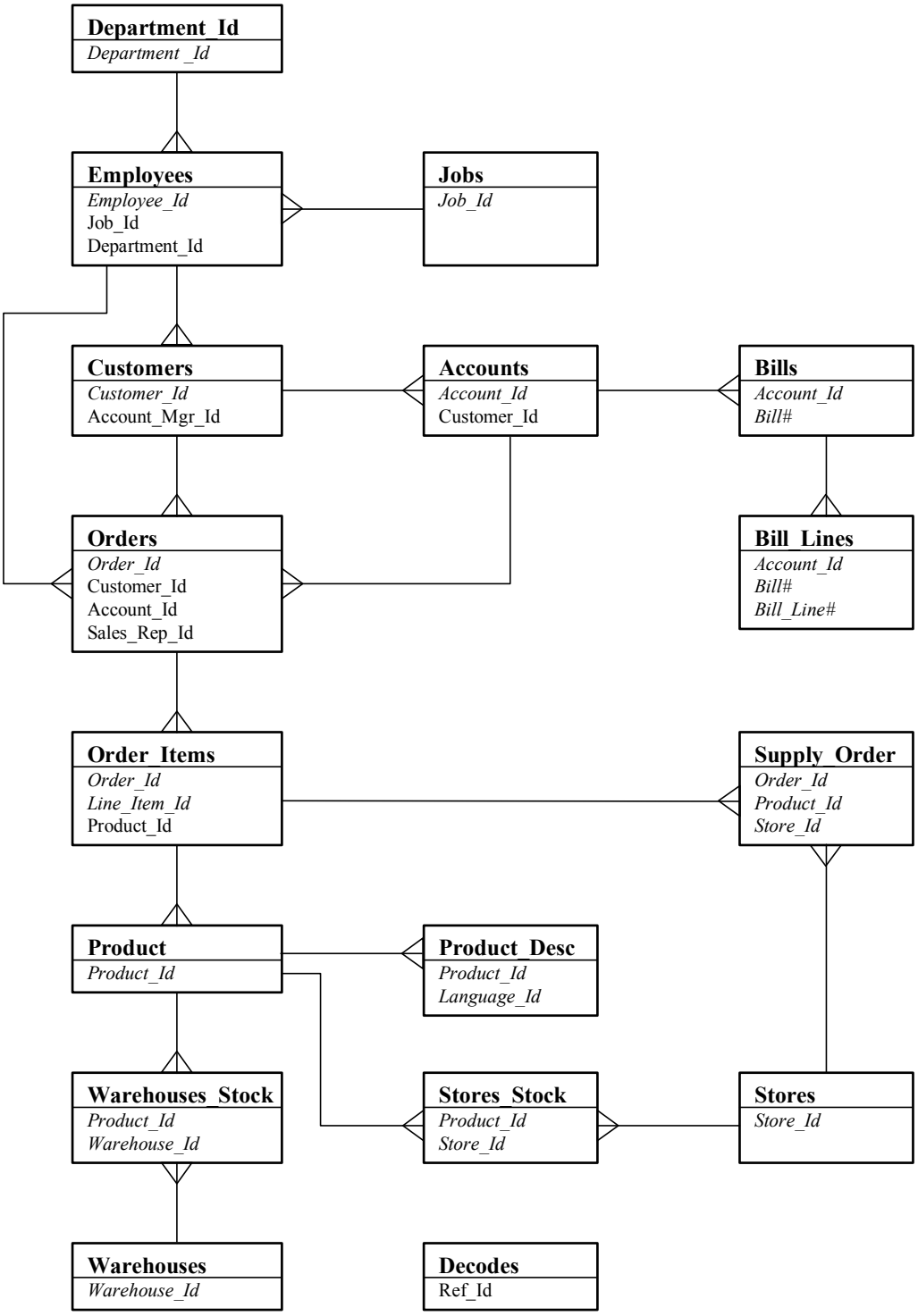
APPENDIX C

Sample Schema

This appendix includes the following information:

- Sample Schema Entity-Relationship diagram
- EXTRACT_KEYS definition

Appendix C 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 *italic*s, 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. There is also a recursive (self-referencing) relationship between EMPLOYEE_ID and MANAGER_ID on table EMPLOYEES that is not shown.

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)

APPENDIX D

Creating an Empty Database

This appendix includes the following information:

- Creating a database
- Creating database structures in the new database

Appendix D Creating an Empty Database

Detailed below are the outline steps for creating a small, empty Oracle database that contains all the elements of the source production database but none of the data. (This is a method that will work for most systems, there are of course many other methods.)

Step 1 - Create a new Oracle database using the Database Creation Assistant (DBCA). Make sure you have only one "data" tablespace, i.e. four tablespaces in total: system, temp, rollback/undo and "data". Specify the "data" tablespace as locally managed, with a uniform extent size of 64 KB and 1 GB in size. This should be should be more than adequate for most systems.

Step 2 - Create user(s) in the new database. Create the Oracle user(s) in the new database with the same name(s) as in the production database and "data" as the default tablespace. Give each of these users the same privileges as they have in the production database and unlimited quota on the "data" tablespace.

```
ALTER USER <user> QUOTA UNLIMITED ON <data_tablespace>;
```

Step 3 - Export required schema(s) from the production database. Use the export utility to create a dump file for each user with all the objects (tables, indexes, stored procedures etc.).

```
$ exp <user>/<passwd> file=<file> rows=no owner=<user>
buffer=100000
```

Where:

`<user>` - user whose data you wish to export
`<passwd>` - password for `<user>`
`<file>` - name of the output export dump file

Step 4 - Import the dump file(s) into the new empty database. Use the import utility to import each dump file, one for each user:

```
$ imp <user>/<passwd> file=<file> buffer=100000
```

Where:

`<user>` - user whose data you wish to import
`<passwd>` - password for `<user>`
`<file>` - name of the export dump file

It is of course possible to use just a single export, and associated import, if you can use a user with the appropriate privileges (IMP_FULL_DATABASE, EXP_FULL_DATABASE). Starting with Oracle 10g you can also use Data Pump (*expdp* and *impdp*) in place of export and import. Data Pump import allows you to perform the import process from the empty target database across a database link using the *network_link* parameter. This removes the need for Step 3, the export.

APPENDIX E

Limitations and Restrictions

This appendix includes the following information:

- Oracle data types

Appendix E Limitations and Restrictions

Oracle Data Types

SelExL version 1.2 supports the following data types:

VARCHAR2	DATE	LONG
RAW	LONG RAW	NUMBER
CHAR	NCHAR	NVARCHAR2
CLOB	BLOB	TIMESTAMP
TIMESTAMP WITH TIMEZONE	TIMESTAMP WITH LOCAL TIMEZONE	

For any additions to this of supported data types go to http://www.christallize.com/.

Tables containing columns with a non-supported data type will still be extracted but with the non-supported columns omitted.

SelExL's two stage method using files for intermediate storage will only extract and load a maximum of 1 MB for a LONG or LONG RAW value. This is because SQL*Loader can currently only load a maximum of 1 MB as a single piece into any single column.