odb User Guide

Version 2.3.0

Table of Contents

License Statement

Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Revision History

Version	Date
2.2.0	TBD
2.1.0	May 1, 2017
2.0.1	July 7, 2016
2.0.0	June 6, 2016
1.3.0	January, 2016

Chapter 1. About This Document

This guide describes how to use odb, a multi-threaded, ODBC-based command-line tool, to perform various operations on a Trafodion database.

In the current release of Trafodion, only loading, extracting, and copying data operations are production ready, meaning that that have been fully tested and are ready to be used in a production environment.

Other features are designated as *Technology Preview* meaning that they have not been fully tested and are not ready for production use.

1.1. Intended Audience

This guide is intended for database administrators and other users who want to run scripts that operate on a Trafodion database, primarily for parallel data loading.

1.2. New and Changed Information

This manual guide is new.

1.3. Notation Conventions

This list summarizes the notation conventions for syntax presentation in this manual.

- UPPERCASE LETTERS

  Uppercase letters indicate keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required.

  ```
  SELECT
  ```

- lowercase letters

  Lowercase letters, regardless of font, indicate variable items that you supply. Items not enclosed in brackets are required.

```
file-name
```

- [ ] Brackets

  Brackets enclose optional syntax items.

```
DATETIME [start-field TO] end-field
```

  A group of items enclosed in brackets is a list from which you can choose one item or none.

  The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines.

  For example:

```
DROP SCHEMA schema [CASCADE]
DROP SCHEMA schema [ CASCADE | RESTRICT ]
```

- { } Braces

  Braces enclose required syntax items.

```
FROM { grantee [, grantee ] ... }
```

  A group of items enclosed in braces is a list from which you are required to choose one item.

  The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines.

  For example:

```
INTERVAL { start-field TO end-field }
{ single-field }
INTERVAL { start-field TO end-field | single-field }
```

- | Vertical Line

  A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces.

```
{expression | NULL}
```

- … Ellipsis

  An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times.

```
ATTRIBUTE[S] attribute [, attribute] ...
{, sql-expression } ...
```

  An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times.

  For example:

```
expression-n ...
```

- Punctuation

  Parentheses, commas, semicolons, and other symbols not previously described must be typed as shown.

```
DAY (datetime-expression)
@script-file
```

  Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must type as shown.

  For example:

```
"{" module-name [, module-name] ... "}"
```

- Item Spacing

  Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma.

  ```
  DAY (datetime-expression) DAY(datetime-expression)
  ```

  If there is no space between two items, spaces are not permitted. In this example, no spaces are permitted between the period and any other items:

  ```
  myfile.sh
  ```

- Line Spacing

  If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line.

  This spacing distinguishes items in a continuation line from items in a vertical list of selections.

  ```
  match-value [NOT] LIKE _pattern
     [ESCAPE esc-char-expression]
  ```

1.4. Comments Encouraged

We encourage your comments concerning this document. We are committed to providing documentation that meets your needs. Send any errors found, suggestions for improvement, or compliments to user@trafodion.apache.org.

Include the document title and any comment, error found, or suggestion for improvement you have concerning this document.

Chapter 2. Introduction

2.1. What is odb

odb is a platform independent, multi-threaded, ODBC command-line tool you can use as a:

- Parallel data loader/extractor
- Query driver (Technology Preview)
- SQL interpreter (Technology Preview)

odb is written in ANSI C. Currently, odb is available only in a 64-bit version for the Linux platform, linked to the unixODBC driver manager.

odb executables use the following naming convention, ***odbAABCC***, where:

- **AA** can be 64 (bit) (32 bit is not currently supported).
- **B** identifies the platform/compiler:
- `l` = Linux/gcc
- `w` = Windows/MS Visual Studio (not yet tested)
- **CC** identifies the ODBC Driver Manager to which odb was linked:
- `uo` = unixODBC Driver Manager
- `ms` = Microsoft ODBC Driver Manager (not yet tested)

So, for example:

- **`odb64luo`** is the 64-bit executable for Linux linked with the unixODBC Driver Manager.

This document contains examples run with the **`odb64luo`** executable.

Chapter 3. Installation and Configuration

See the Trafodion Client Installation Guide for install instructions.

Refer to the unixODBC documentation for additional information for `unixODBC`.

Chapter 4. Basic Concepts

4.1. Get Help

The following command shows the odb help:

```
~/Devel/odb $ ./odb64luo -h
odb version 1.3.0
Build: linux, amd64, gcc generic m64, uodbc, mreadline, dynamic gzip, dynamic libhdfs,
dynamic libxml2 [Mar 30 2015 00:29:25]
   -h: print this help
   -version: print odb version and exit
   -lsdrv: list available drivers @ Driver Manager level
   -lsdsn: list available Data Sources
Connection related options. You can connect using either:
   -u User: (default $ODB_USER variable)
   -p Password: (default $ODB_PWD variable)
   -d Data_Source_Name: (default $ODB_DSN variable)
   -ca Connection_Attributes (normally used instead of -d DSN)
   -U sets SQL_TXN_READ_UNCOMMITTED isolation level
   -ndsn [+]<number>: adds 1 to <number> to DSN
   -nps <nbytes>[:<nbytes>]: specify source[:target] network packet size
SQL interpreter options:
   -I [$ODB_INI SECTION]: interactive mode shell
   -noconnect: do not connect on startup General options:
   -q [cmd|res|all|off]: do not print commands/results/both
   -i [TYPE[MULT,WIDE_MULT]:CATALOG.SCHEMA[.TABLE]]: lists following object types:
      (t)ables, (v)iews, s(y)nonyns, (s)chemas, (c)atalogs, syst(e)m tables
      (l)ocal temp, (g)lobal temp, (m)at views, (M)mat view groups, (a)lias
      (A)ll object types, (T)table desc, (D)table DDL, (U) table DDL with multipliers
   -r #rowset: rowset to be used insert/selects (default 100)
   -soe: Stop On Error (script execution/loading task)
   -N : Null run. Doesn't SQLExecute statements
   -v : be verbose
   -vv : Print execution table
   -noschema : do not use schemas: CAT.OBJ instead of CAT.SCH.OBJ
   -nocatalog : do not use catalogs: SCH.OBJ instead of CAT.SCH.OBJ
   -nocatnull : like -nocatalog but uses NULL instead of empty CAT strings
   -ucs2toutf8 : set UCS-2 to UTF-8 conversion in odb
   -var var_name var_value: set user defined variables
   -ksep char/code: Thousands Separator Character (default ',')
   -dsep char/code: Decimal Separator Character (default '.')
SQL execution options [connection required]:
   -x [#inst:]'command': runs #inst (default 1) command instances
   -f [#inst:]'script': runs #inst (default 1) script instances
   -P script_path_regexp: runs in parallel scripts_path_regexp if script_path_regexp
ends with / all files in that dir
   -S script_path_regexp: runs serially scripts_path_regexp if script_path_regexp ends
with / all files in that dir
   -L #loops: runs everything #loops times
```

```
   -T max_threads: max number of execution threads
   -dlb: use Dynamic Load Balancing
   -timeout #seconds: stops everything after #seconds (no Win32)
   -delay #ms: delay (ms) before starting next thread
   -ldelay #ms: delay (ms) before starting next loop in a thread
   -ttime #ms[:ms]: delay (ms) before starting next command in a thread random delay if
a [min:max] range is specified
   -F #records: max rows to fetch
   -c : output in csv format
   -b : print start time in the headers when CSV output
   -pcn: Print Column Names
   -plm: Print Line Mode
   -fs char/code: Field Sep <char> ASCII_dec> 0<ASCII_OCT> X<ASCII_HEX>
   -rs char/code: Rec Sep <char> ASCII_dec> 0<ASCII_OCT> X<ASCII_HEX>
   -sq char/code: String Qualifier (default none)
   -ec char/code: Escape Character (default '\')
   -ns nullstring: print nullstring when a field is NULL
   -trim: Trim leading/trailing white spaces from txt cols
   -drs: describe result set (#cols, data types...) for each Q)
   -hint: do not remove C style comments (treat them as hints)
   -casesens: set case sensitive DB
   -Z : shuffle the execution table randomizing Qs start order
Data loading options [connection required]:
   -l src=[-]file:tgt=table[:map=mapfile][:fs=fieldsep][:rs=recsep][:soe]
      [:skip=linestoskip][:ns=nullstring][:ec=eschar][:sq=stringqualifier]
      [:pc=padchar][:em=embedchar][:errmax=#max_err][:commit=auto|end|#rows|x#rs]
      [:rows=#rowset][:norb][:full][:max=#max_rec][:truncate][:show][:bpc=#][:bpwc=#]
      [:nomark][:parallel=number][:iobuff=#size][:buffsz=#size]][:fieldtrunc={0-4}]
      [:pre={@sqlfile}|{[sqlcmd]}][:post={@sqlfile}|{[sqlcmd]}][:ifempty]

[:direct][:bad=[+]badfile][:tpar=#tables][:maxlen=#bytes][:time][:loadcmd=IN|UP|UL]
      [:xmltag=[+]element][:xmlord][:xmldump]
      Defaults/notes:
      * src file: local file or {hdfs,mapr}[@host,port[,huser]].<HDFS_PATH>
      * fs: default ','. Also <ASCII_dec> 0<ASCII_OCT> X<ASCII_HEX>
      * rs: default '\n'. Also <ASCII_dec> 0<ASCII_OCT> X<ASCII_HEX>
      * ec: default '\'. Also <ASCII_dec> 0<ASCII_OCT> X<ASCII_HEX>
      * pc: no default. Also <ASCII_dec> 0<ASCII_OCT> X<ASCII_HEX>
      * direct: only for Vertica databases
      * bpc: default 1,bpwc: default 4
      * loadcmd: default IN. only for {project-name} databases
Data extraction options [connection required]:
   -e {src={table|-file}|sql=<custom sql>}:tgt=[+]file[:pwhere=where_cond]
      [:fs=fieldsep][:rs=recsep][:sq=stringqualifier][:ec=escape_char][:soe]

[:ns=nullstring][es=emptystring][:rows=#rowset][:nomark][:binary][:bpc=#][:bpwc=#]
      [:max=#max_rec][:[r]trim[+]][:cast][:multi][parallel=number][:gzip[=lev]]
      [:splitby=column][:uncommitted][:iobuff=#size][hblock=#size][:ucs2toutf8]

[:pre={@sqlfile}|{[sqlcmd]}[:mpre={@sqlfile}|{[sqlcmd]}[:post={@sqlfile}|{[sqlcmd]}]
      [tpar=#tables][:time][:cols=[-]columns]][:maxlen=#bytes][:xml]
      Defaults/notes:
      * tgt file: local file or {hdfs,mapr}.[@host,port[,huser]].<HDFS_PATH>
      * fs: default ','. Also <ASCII_dec> 0<ASCII_OCT> X<ASCII_HEX>
```

```
       * rs: default '\n'. Also <ASCII_dec> 0<ASCII_OCT> X<ASCII_HEX>
       * ec: default '\'. Also <ASCII_dec> 0<ASCII_OCT> X<ASCII_HEX>
       * sq: no default. Also <ASCII_dec> 0<ASCII_OCT> X<ASCII_HEX>
       * gzip compression level between 0 and 9
       * bpc: default 1,bpwc: default 4
Data copy options [connection required]:
   -cp src={table|-
file:tgt=schema[.table][pwhere=where_cond][:soe][:roe=#][:roedel=#ms]
       [:truncate][:rows=#rowset][:nomark][:max=#max_rec][:bpc=#][:bpwc=#][:[r]trim[+]]
       [:parallel=number][errmax=#max_err][:commit=auto|end|#rows|x#rs][:time] ][:cast]
       [:direct][:uncommitted][:norb][:splitby=column][:pre={@sqlfile}|{[sqlcmd]}]
       [:post={@sqlfile}|{[sqlcmd]}][:mpre={@sqlfile}|{[sqlcmd]}][:ifempty]
       [:loaders=#loaders][:tpar=#tables][:cols=[-]columns][:errdmp=file]
][:loadcmd=IN|UP|UL]
       [sql={[sqlcmd]|@sqlfile|-file}[:bind=auto|char|cdef][:seq=field#[,start]]
       [tmpre={@sqlfile}|{[sqlcmd]}][:ucs2toutf8=[skip,force,cpucs2,qmark]]
       Defaults/notes:
       * loaders: default 2 load threads for each 'extractor'
       * direct: only work if target database is Vertica
       * ucs2toutf8: default is 'skip'
       * roe: default 3 if no arguments
       * bpc: default 1,bpwc: default 4
       * loadcmd: default IN. only for {project-name} databases
Data pipe options [connection required]:
   -pipe sql={[sqlcmd]|@sqlscript|-file}:tgtsql={@sqlfile|[sqlcmd]}[:soe]
      [:rows=#rowset][:nomark][:max=#max_rec][:bpc=#][:bpwc=#][:errdmp=file]
      [:parallel=number][errmax=#max_err][:commit=auto|end|#rows|x#rs][:time]
      [:pre={@sqlfile}|{[sqlcmd]}][:post={@sqlfile}|{[sqlcmd]}]
      [:mpre={@sqlfile}|{[sqlcmd]}][:tmpre={@sqlfile}|{[sqlcmd]}]
      [:loaders=#loaders][:tpar=#tables][:bind=auto|char|cdef]
      Defaults/notes:
      * loaders: default 1 load threads for each extraction thread
      * bpc: default 1,bpwc: default 4
```

```
Table diff options [connection required]:
   -diff src={table|-file}:tgt=table:[key=columns][:output=[+]file][:pwhere=where_cond]
      [:pwhere=where_cond][:nomark][:rows=#rowset][:odad][:fs=fieldsep][:time][trim[+]]
      [:rs=recsep][:quick][:splitby=column][:parallel=number][:max=#max_rec]

[:print=[I][D][C]][:ns=nullstring][:es=emptystring][:bpc=#][:bpwc=#][:uncommitted]
      [:pre={@sqlfile}|{[sqlcmd]}][:post={@sqlfile}|{[sqlcmd]}][tpar=#tables]
      Defaults/notes:
      * bpc: default 1,bpwc: default 4
      * print: default is Inserted Deleted Changed
```

4.2. Connect to Database

odb uses standard ODBC APIs to connect to a database.

Normally you have to provide the following information: user, password and ODBC data source.

Example

```
$ ./odb64luo –u user –p password –d dsn ...
```

You can provide Driver-specific connection attributes using th `-ca` command line option.

Command-line passwords are protected against `ps -ef` sniffing attacks under *nix. You can safely pass your password via `–p`. An alternative approach is to use environment variables or the odb password prompt (see below).

odb will use the following environment variables (if defined):

Variable	Meaning	Corresponding Command-Line Option
`ODB_USER`	User name to use for database connections	`-u <user>`
`ODB_PWD`	Password for database connections	`-p <passwd>`
`ODB_DSN`	DSN for database connection	`-d <dsn>`
`ODB_INI`	Init file for interactive shell	
`ODB_HIST`	history file name to save command history on exit	

Command-line options take precedence over environment variables.

4.3. List ODBC Drivers and Data Sources

You can list available drivers with `-lsdrv`:

```
~/Devel/odb $ ./odb64luo -lsdrv
Trafodion - Description=Trafodion ODBC Stand Alone Driver
...
```

You can list locally configured data sources with `-lsdsn`:

```
~/Devel/odb $ ./odb64luo -lsdsn
traf - Trafodion
VMFELICI – Vertica
...
```

4.4. Obtain Database Information

The `-i` option allows you to get information about the database you're connecting to as well as the ODBC driver. It's a simple way to check your credentials and database connection.

Example

```
~/mauro/odb $ ./odb64luo -u xxx -p xxx -d traf -i

odb [2015-04-20 21:20:47]: starting ODBC connection(s)... 0
       [odb version 1.3.0]
       Build: linux, amd64, gcc generic m64, uodbc, mreadline, dynamic gzip, dynamic
libhdfs, dynamic libxml2 [Apr 8 2015 16:47:49]

       DBMS product name (SQL_DBMS_NAME)            : Trafodion
       DBMS product version (SQL_DBMS_VER)          : 01.03.0000
       Database name (SQL_DATABASE_NAME)            : TRAFODION
       Server name (SQL_SERVER_NAME)                : --name--
       Data source name (SQL_DATA_SOURCE_NAME)      : traf
       Data source RO (SQL_DATA_SOURCE_READ_ONLY)   : N
       ODBC Driver name (SQL_DRIVER_NAME)           : libhpodbc64.so
       ODBC Driver version (SQL_DRIVER_VER)         : 03.00.0000
       ODBC Driver level (SQL_DRIVER_ODBC_VER)      : 03.51
       ODBC Driver Manager version (SQL_DM_VER)     : 03.52.0002.0002
       ODBC Driver Manager level (SQL_ODBC_VER)     : 03.52
       Connection Packet Size (SQL_ATTR_PACKET_SIZE): 0
odb [2015-04-20 21:20:48]: exiting. Session Elapsed time 0.229 seconds (00:00:00.229)
```

4.5. List Database Objects

The previous section used the `-i` option without any argument.

This option accepts arguments with the following syntax:

```
[TYPE:][CATALOG.SCHEMA][.OBJECT]
```

where type can be:

Type	Meaning
`<missing>`	All database object types
`A:`	All database object types
`t:`	Tables
`v:`	Views
`a:`	Aliases
`y:`	Synonyms
`l:`	Local Temporary
`g:`	Global Temporary
`m:`	Materialized views
`M:`	Materialized view groups
`s:`	Schemas
`c:`	Catalogs
`T:`	Table descriptions
`D:`	Table DDL
`U[x,y]:`	Table DDL multiplying wide columns by Y and non-wide columns by X

Example	Action
`-i c:`	List all catalogs.
`-i s:`	List all schemas.
`-i TRAFODION.MFTEST`	List all objects in `TRAFODION.MFTEST` schema.
`-i t:TRAFODION.MFTEST`	List all tables in `TRAFODION.MFTEST`.
`-i t:TRAFODION.MFTEST.A%`	List all tables in `TRAFODION.MFTEST` schema staring with `A`.
`-i v:TRAFODION.MFTEST`	List all views in `TRAFODION.MFTEST`.
`-i v:TRAFODION.MFTEST.%_V`	List all views in `TRAFODION.MFTEST` ending with `_V`.
`-i T:TRAFODION.MFTEST.STG%`	Describe all tables starting with `STG` in `TRAFODION.MFTEST`.

Extended Examples

```
~/mauro/odb $ ./odb64luo -u MFELICI -p xxx -d MFELICI -i T:TRAFODION.MAURIZIO.T%

odb [2011-12-07 14:43:51]: starting (1) ODBC connection(s)... 1
Describing: TRAFODION.MAURIZIO.T1
+------+--------------+----+-------+-------+
|COLUMN|TYPE          |NULL|DEFAULT|INDEX  |
+------+--------------+----+-------+-------+
|ID    |INTEGER SIGNED|YES |       |       |
|NAME  |CHAR(10)      |YES |       |       |
|LASTN |VARCHAR(20)   |YES |       |       |
+------+--------------+----+-------+-------+
Describing: TRAFODION.MAURIZIO.T11
+------+--------------+----+-------+-------+
|COLUMN|TYPE          |NULL|DEFAULT|INDEX  |
+------+--------------+----+-------+-------+
|ID    |INTEGER SIGNED|NO  |       |T11 1 U|
|NAME  |CHAR(10)      |YES |       |       |
+------+--------------+----+-------+-------+
```

The `INDEX` column (when using type `T`) contains the following information:

- `name` of the `INDEX` (in Trafodion indexes having the same name as the table are Primary Keys).
- `ordinal number` to identify the order of that field in the index.
- `(U)nique o (M)ultiple` values allowed.
- `(+)` means that more than one index includes that field.

4.6. Perform Actions on Multiple Database Objects

odb uses extended SQL syntax to execute actions on multiple objects: `&<type>:<path>` - where `<type>` is one of the object types listed in the previous section.

Example

Example	Action
`delete from &t:MF%`	Purge ALL tables (t:) staring with `M”`.
`drop view &v:mftest.%vw`	Drop ALL views (v:) ending with `_VW` in the schema `MFTEST`.
`UPDATE STATISTICS FOR TABLE &t:TRAFODION.MFTEST.%`	Update Stats for ALL tables in `TRAFODION.MFTEST`.

You can use this *extended* SQL syntax in the SQL Interpreter or generic SQL scripts.

4.7. Run Commands and Scripts

The `–x` switch can be used to run generic SQL commands. You can also use `–f` to run SQL scripts:

1. `-x "SQL command"` to run a specific SQL command.
2. `-f <script>` to run a script file.

Example

```
~/Devel/odb $ ./odb64luo -x "select count(*) from customer"

150000
[0.0.0]--- 1 row(s) selected in 0.137s (prep 0.000s, exec 0.137s, 1st fetch 0.000s,
fetch 0.000s)
```

The meaning of `[0.0.0]` will be explained later.

```
~/Devel/odb $ cat script.sql

SELECT COUNT(*) FROM T1;
-- This is a comment
SELECT
        L_RETURNFLAG
      , L_LINESTATUS
      , SUM(L_QUANTITY) AS SUM_QTY
      , SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE
      , SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS SUM_DISC_PRICE
      , SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE
      , AVG(L_QUANTITY) AS AVG_QTY
      , AVG(L_EXTENDEDPRICE) AS AVG_PRICE
      , AVG(L_DISCOUNT) AS AVG_DISC
      , COUNT(*) AS COUNT_ORDER
FROM
        LINEITEM
WHERE
        L_SHIPDATE <= DATE '1998-12-01' - INTERVAL '90' DAY
GROUP BY
        L_RETURNFLAG, L_LINESTATUS
ORDER BY
        L_RETURNFLAG, L_LINESTATUS
;
```

```
~/Devel/odb $ ./odb64luo -f script.sql

[0.0.0]Executing: 'SELECT COUNT(*) FROM T1;'
5
[0.0.0]--- 1 row(s) selected in 0.015s (prep 0.000s, exec 0.015s, 1st fetch -0.000s,
fetch -0.000s)
[0.0.1]Executing: 'SELECT L_RETURNFLAG, L_LINESTATUS, SUM(L_QUANTITY) AS SUM_QTY,
SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS
SUM_DISC_PRICE, SUM(L_EXTENDEDPRICE*(1- L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE,
AVG(L_QUANTITY) AS AVG_QTY, AVG(L_EXTENDEDPRICE) AS AVG_PRICE, AVG(L_DISCOUNT) AS
AVG_DISC, COUNT(*) AS COUNT_ORDER FROM LINEITEM WHERE L_SHIPDATE <= DATE '1998-12-01'
- INTERVAL '90' DAY GROUP BY L_RETURNFLAG, L_LINESTATUS ORDER BY L_RETURNFLAG,
L_LINESTATUS;'
A,F,37734107.00,56586554400.73,53758257134.8700,55909065222.827692,25.522006,
38273.129735,0.049985,1478493
...
R,F,37719753.00,56568041380.90,53741292684.6040,55889619119.831932,25.505794,
38250.854626,0.050009,1478870
[0.0.1]--- 4 row(s) selected in 21.344s (prep 0.000s, exec 21.344s, 1st fetch 0.000s,
fetch 0.000s)
```

You can use the `-q` switch to omit selected output components.

Example

`–q` cmd will not print the ***commands*** being executed:

```
~/Devel/odb $ ./odb64luo -f script.sql –q cmd

5
[0.0.0]--- 1 row(s) selected in 0.015s (prep 0.000s, exec 0.015s, 1st fetch -0.000s,
fetch -0.000s)

A,F,37734107.00,56586554400.73,53758257134.8700,55909065222.827692,25.522006,
38273.129735,0.049985,1478493
...
R,F,37719753.00,56568041380.90,53741292684.6040,55889619119.831932,25.505794,
38250.854626,0.050009,1478870
[0.0.1]--- 4 row(s) selected in 21.344s (prep 0.000s, exec 21.344s, 1st fetch 0.000s,
fetch 0.000s)
```

While `-q res` will nit print the ***results***:

```
~/Devel/odb $ ./odb64luo -f script.sql –q res

[0.0.0]Executing: 'SELECT COUNT(*) FROM T1;'
[0.0.0]--- 1 row(s) selected in 0.015s (prep 0.000s, exec 0.015s, 1st fetch -0.000s,
fetch -0.000s)
[0.0.1]Executing: 'SELECT L_RETURNFLAG,L_LINESTATUS, SUM(L_QUANTITY) AS
SUM_QTY, SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT))
AS SUM_DISC_PRICE, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE,
AVG(L_QUANTITY) AS AVG_QTY, AVG(L_EXTENDEDPRICE) AS AVG_PRICE, AVG(L_DISCOUNT) AS
AVG_DISC, COUNT(*) AS COUNT_ORDER FROM LINEITEM WHERE L_SHIPDATE <= DATE '1998-12-01'
- INTERVAL '90' DAY GROUP BY L_RETURNFLAG, L_LINESTATUS ORDER BY L_RETURNFLAG,
L_LINESTATUS;'
[0.0.1]--- 4 row(s) selected in 21.344s (prep 0.000s, exec 21.344s, 1st fetch 0.000s,
fetch 0.000s)
```

`-q all` (or just `-q`) will not print neither the ***commands*** nor the ***results***:

```
~/Devel/odb $ ./odb64luo -f script.sql -q all

[0.0.0]--- 1 row(s) selected in 0.015s (prep 0.000s, exec 0.015s, 1st fetch -0.000s,
fetch -0.000s)
[0.0.1]--- 4 row(s) selected in 21.344s (prep 0.000s, exec 21.344s, 1st fetch 0.000s,
fetch 0.000s)
```

This is often used with odb as query driver.

Even when odb doesn't print query results (`-q res`), the result set will be fetched and data is transferred from the database server to the client. In other words, `-q res` is somehow similar (but not exactly equivalent) to a `/dev/null` output redirection.

A special file name you can use with `-f` is `-` (dash).

It means: read the script to be executed from the *standard input*.

Example

The following command will *copy* table definitions from one system to another recreating, on the target system, the same table structures as in the source system:

```
$ odb64luo –u u1 -p p1 -d SRC -i t:TRAFODION.CIV04 -x "SHOWDDL &1" \
| odb64luo –u u2 –p p2 -d TGT -f -
```

4.8. Shell Script "here document" Syntax

Commonly, there's a need to *embed* SQL commands in shell scripts.

Use the `-f -` (read commands from standard input) odb syntax.

Example

```
odb64luo -f - <<-EOF 2>&1 | tee -a $\{LOG}
   drop table &t:TRAFODION.maurizio.ml%;
   create table ml2
   (
     id integer
   , fname char(10)
   , bdate date
   , lname char(10) default 'Felici'
   , comment char(20)
   , city char(10)
   ) no partitions;
EOF
```

4.9. Parallelize Multiple Commands and Scripts

odb uses threads to run multiple commands in parallel. Each command (`-x`) or script (`-f`) will be executed, independently from the others, using a different thread.

Example

Running scripts in parallel.

```
~/Devel/odb $ ./odb64luo -x "select count(*) from types" -f script1.sql
```

Uses two *independent* threads executed in parallel. The first thread will run `select count(*) from types` and the other `script1.sql`.

You can also run ***multiple copies*** of the same command by adding `<num>:` before `-x` or `-f` arguments.

The following command runs the instances of `select count(*) from types `, five instances of `script1.sql` and three instances of `script2.sql` in parallel using `3 + 5 + 3 = 11` threads in total:

Example

Running eleven commands and scripts in parallel

```
~/Devel/odb $ ./odb64luo -x 3:"select count(*) from types" -f 5:script1.sql \
-f 3:script2.sql -q

[1.0.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[0.0.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[2.0.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[4.0.0]--- 1 row(s) selected in 0.001s (prep 0.000s, exec 0.001s, fetch 0.000s/0.000s)
[6.0.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[5.0.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[3.0.0]--- 1 row(s) selected in 0.001s (prep 0.000s, exec 0.001s, fetch 0.000s/0.000s)
[8.0.0]--- 1 row(s) selected in 0.001s (prep 0.000s, exec 0.001s, fetch 0.000s/0.000s)
[7.0.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[9.0.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[10.0.0]--- 1 row(s) selected in 0.001s prep 0.000s, exec 0.001s, fetch 0.000s/0.000s)
```

The first number in `[1.0.0]` is the ***thread ID***. Thread IDs are assigned by odb starting from zero.

You can limit the maximum number of threads with `-T` option.

Example

The following command runs the same 11 commands/scripts limiting the number of threads (**and ODBC connections**) to 4:

```
~/Devel/odb $ ./odb64luo -x 3:"select count(*) from types" -f 5:script1.sql \
-f 3:script2.sql -q -T 4

[1.0.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[0.0.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[2.0.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[1.3.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[2.1.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[0.1.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[3.0.0]--- 1 row(s) selected in 0.001s (prep 0.000s, exec 0.001s, fetch 0.000s/0.000s)
[2.2.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[3.1.0]--- 1 row(s) selected in 0.001s (prep 0.000s, exec 0.001s, fetch 0.000s/0.000s)
[0.2.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
[1.2.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, fetch 0.000s/0.000s)
```

The thread ID is now in the 0-3 range because the 11 ***executions*** were ***queued*** into four threads. odb offers several alternatives to queue `M` executions in `N(<M)` threads. See below.

4.10. Limit Number of odb Threads

By default, odb creates as many threads as the numbers of executions.

The command in the following example creates `1 + 3 + 3 = 7` threads. Each thread will start its own ODBC connection.

Example

```
~/src/C/odb $ ./odb64luo -f script1.sql -f 3:script2.sql -x 3:"<mysqlcmd>"
```

You can limit the max number of threads using `–T`.

Example

```
~/src/C/odb $ ./odb64luo -f script1.sql -f 3:script2.sql -x 3:"<mysqlcmd>" -T 2
```

This command creates just two threads to execute the seven commands/scripts. odb will never create more threads than needed:

Example

```
~/Devel/odb $ ./odb64luo -f 2:script1.sql -f 3:script2.sql -T 8 -c -q odb [main(1017)] - Warning: won’t be created more thread (8) than needed (5).
```

4.11. Change Executions Distributed Across Threads

By default, executions are distributed in round-robin across threads.

Example

```
~/src/C/odb $ ./odb64luo -f script1.sql -f 3:script2.sql -x 3:"<mysqlcmd>" -T 3
```

Using the command above, the execution queue will be as follows:

	Thread 1	Thread 2	Thread3
Third Execution	`mysqlcmd`		
Second Execution	`Script2.sql`	`mysqlcmd`	`mysqlcmd`
First Execution	`Script1.sql`	`Script2.sql`	`Script2.sql`

This (standard) behavior can be modified using the following options:

- `-Z` (shuffle): This option ***randomizes*** the execution order.
- ***factor sign*** with `-P` option: See Run All Scripts With a Given Path.
- `-dlb` (Dynamic Load Balancing): See Dynamic Load Balancing.

4.12. Dynamic Load Balancing

As discussed in the previous section, executions are normally *pre-assigned* to threads using a simple round-robin algorithm. This way, the total elapsed time for each thread depends on the complexity of **its own** *executions*.

Example

Suppose you have two threads and two *executions* per thread:

	Thread 1	Thread 2
Second Execution	`Script1.2`	`Script2.2`
First Execution	`Script1.3`	`Script2.1`

If thread 2.1 and 2.2 require a very short time to be executed you can have a situation where Thread2 has nothing to do (it will be terminated) while Thread1 is still busy with **its own** Script1.3 and Script1.2.

In some cases, for example during data extractions (see Load Binary Files), you might want to keep all threads busy at any given time. In these cases you can use Dynamic Load Balancing (`-dlb`). With Dynamic Load Balancing jobs are not **pre-assigned** to threads when odb starts; each thread will pick the next job to run from the job list *at run-time*.

4.13. Use Variables in odb Scripts

odb let you to use two kinds of variables:

- **Internal Variables** defined through the `set param` command and identified by the ampersand character;
- **Environment variables** defined at operating system level and identified by a dollar sign;

You can mix internal and environment variables in your scripts. If a variable is not expanded to a valid Internal/Environment variable the text will remain unchanged.

```
~/Devel/odb $ cat scr.sql set param region1 ASIA

-- region1 is defined as an internal odb parameter
select * from tpch.region where r_name = '&region1';
-- region2 is defined as an environment variable
select * from tpch.region where r_name = '$region2';
-- you can mix internal and environment variables
select * from tpch.region where r_name = '$region2' or r_name = '&region1';
-- region3 variable does not exists so it won't be not expanded
select * from tpch.region where r_name = '&region3';
```

After you define `region2` at operating system level:

```
~/Devel/odb $ export region2=AMERICA
```

Output:

```
~/Devel/odb $ ./odb64luo -u mauro -p xx -d pglocal -f scr.sql

odb [2011-12-12 08:01:31]: starting (1) ODBC connection(s)... 1 [0.0.0]Executing:
'select * from tpch.region where r_name = 'ASIA';' 2,ASIA,ges. thinly even pinto beans
ca
[0.0.0]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, 1st fetch 0.000s,
fetch 0.000s)
[0.0.1]Executing: 'select * from tpch.region where r_name = 'AMERICA';' 1,AMERICA,hs
use ironic, even requests. s
[0.0.1]--- 1 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, 1st fetch 0.000s,
fetch 0.000s)
[0.0.2]Executing: 'select * from tpch.region where r_name = 'AMERICA' or r_name =
'ASIA';' 1,AMERICA,hs use ironic,
even requests.s2,ASIA,ges. thinly even pinto beans ca
[0.0.2]--- 2 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, 1st fetch 0.000s,
fetch 0.000s)
[0.0.3]Executing: 'select * from tpch.region where r_name = '&region3';'
[0.0.3]--- 0 row(s) selected in 0.000s (prep 0.000s, exec 0.000s, 1st fetch 0.000s,
fetch 0.000s)
```

4.14. Thread ID, Thread Execution#, and Script Command#

Consider a script containing two commands:

```
~/odb $ cat script.sql

SELECT COUNT(*) FROM ORDERS;
SELECT COUNT(*) FROM SUPPLIER;
```

Running this script ten times using two threads yields:

```
~/odb $ ./odb64luo -f 10:script.sql -q -T 2

[0.0.0]--- 1 row(s) selected in 0.102s (prep 0.022s, exec 0.044s, 1st fetch 0.037s,
fetch 0.037s)
[1.0.0]--- 1 row(s) selected in 0.125s (prep 0.022s, exec 0.068s, 1st fetch 0.036s,
fetch 0.036s)
[0.0.1]--- 1 row(s) selected in 0.520s (prep 0.022s, exec 0.048s, 1st fetch 0.450s,
fetch 0.450s)
[1.0.1]--- 1 row(s) selected in 0.564s (prep 0.017s, exec 0.480s, 1st fetch 0.067s,
fetch 0.067s)
[0.1.0]--- 1 row(s) selected in 0.131s (prep 0.022s, exec 0.060s, 1st fetch 0.048s,
fetch 0.048s)
[0.1.3]--- 1 row(s) selected in 0.086s (prep 0.022s, exec 0.057s, 1st fetch 0.007s,
fetch 0.007s)
[1.3.0]--- 1 row(s) selected in 0.136s (prep 0.035s, exec 0.058s, 1st fetch 0.042s,
fetch 0.042s)
[0.2.0]--- 1 row(s) selected in 0.123s (prep 0.029s, exec 0.068s, 1st fetch 0.026s,
fetch 0.026s)
[1.3.1]--- 1 row(s) selected in 0.119s (prep 0.016s, exec 0.082s, 1st fetch 0.021s,
fetch 0.021s)
[0.2.1]--- 1 row(s) selected in 0.089s (prep 0.031s, exec 0.054s, 1st fetch 0.004s,
fetch 0.004s)
[1.2.0]--- 1 row(s) selected in 0.138s (prep 0.023s, exec 0.041s, 1st fetch 0.073s,
fetch 0.073s)
[0.3.0]--- 1 row(s) selected in 0.144s (prep 0.038s, exec 0.045s, 1st fetch 0.061s,
fetch 0.061s)
[1.2.1]--- 1 row(s) selected in 0.127s (prep 0.016s, exec 0.041s, 1st fetch 0.070s,
fetch 0.070s)
[0.3.1]--- 1 row(s) selected in 0.136s (prep 0.033s, exec 0.056s, 1st fetch 0.048s,
fetch 0.048s)
[1.3.0]--- 1 row(s) selected in 0.131s (prep 0.023s, exec 0.037s, 1st fetch 0.071s,
fetch 0.071s)
[0.4.0]--- 1 row(s) selected in 0.111s (prep 0.033s, exec 0.045s, 1st fetch 0.033s,
fetch 0.033s)
[0.4.1]--- 1 row(s) selected in 0.076s (prep 0.033s, exec 0.037s, 1st fetch 0.005s,
fetch 0.006s)
[1.3.1]--- 1 row(s) selected in 0.098s (prep 0.016s, exec 0.065s, 1st fetch 0.017s,
fetch 0.017s)
[1.4.0]--- 1 row(s) selected in 0.133s (prep 0.023s, exec 0.074s, 1st fetch 0.035s,
fetch 0.035s)
[1.4.1]--- 1 row(s) selected in 0.098s (prep 0.017s, exec 0.064s, 1st fetch 0.016s,
fetch 0.016s)
```

The numbers between square brackets have the following meaning:

1. The first digit is the ***thread ID***. The example above has two threads; the ID is either 0 or 1.
2. The second digit is the ***execution#*** for a given thread. The example above has ten script executions for two threads, each thread will have to execute this script five times.; ***execution#***, is between 0 and 4.
3. The third (last) digit is the ***command#*** in a given script. The script in the example above contains two commands; this value is 0 or 1.

Example

`[0.3.1]` means that the **first thread** (*thread id=0*) was executing its **fourth job** (*thread execution#=3*) and — more specifically &8212; the **second command** in that script (*script command=1*).

4.15. Validate SQL Scripts

You can check commands and SQL scripts with odb using the `-N` (null run) flag. This will just `prepare` (compile) the commands without executing them and fetching the results.

4.16. Different Data Sources for Different Threads

Normally all ODBC connections started by odb will use the same Data Source. However, there could be special cases where you want to use different DSN for different threads. In these cases you can use the `–ndsn <number>` option. This will append to the Data Source name specified via `–d` a suffix from `1` to `-ndsn` argument.

Example

```
$ ./odb64luo ... –d MYDSN –ndsn 4
```

It will use the following (round-robin) DSN/thread association: `MYDSN1` for the first thread, `MYDSN2` for the second thread and so on. The fifth thread (if any) will use `MYDSN1` again. You can use a sequential DSN/thread association by using a `+` sign in front of the `–ndsn` argument.

Example

If you have 16 threads and `–d MYDSN`:

Thread ID	**DSN with** `–ndsn 8`	**DSN with** `–ndsn +8`
0	MYDSN1	MYDSN1
1	MYDSN2	MYDSN1
2	MYDSN3	MYDSN2
3	MYDSN4	MYDSN2
4	MYDSN5	MYDSN3
5	MYDSN6	MYDSN3
6	MYDSN7	MYDSN4
7	MYDSN8	MYDSN4
8	MYDSN1	MYDSN5
9	MYDSN2	MYDSN5
10	MYDSN3	MYDSN6
11	MYDSN4	MYDSN6
12	MYDSN5	MYDSN7
13	MYDSN6	MYDSN7
14	MYDSN7	MYDSN8
15	MYDSN8	MYDSN8

This technique has been used to maximize extraction throughput from a multi-segment Trafodion system. Each (local)

Data Source was **linked** to a corresponding remote Data Source extracting its own data through its own network interface card.

4.17. Format Query Results

Normally odb prints query results using a very basic CSV format.

Example

```
$ ./odb64luo -x "select s_suppkey, s_name, s_phone from tpch.supplier limit 5

1,Supplier#000000001,27-918-335-1736
2,Supplier#000000002,15-679-861-2259
3,Supplier#000000003,11-383-516-1199
4,Supplier#000000004,25-843-787-7479
5,Supplier#000000005,21-151-690-3663
```

Adding the option `-pad` you generates the output in table format:

```
$ ./odb64luo -x "select s_suppkey, s_name, s_phone from tpch.supplier limit 5" -pad

s_suppkey      |s_name                    |s_phone
---------------+--------------------------+---------------
1              |Supplier#000000001        |27-918-335-1736
2              |Supplier#000000002        |15-679-861-2259
3              |Supplier#000000003        |11-383-516-1199
4              |Supplier#000000004        |25-843-787-7479
5              |Supplier#000000005        |21-151-690-3663
```

4.18. Extract Table DDL

You can extract DDL from one or several tables using either the `-i D…` or `-i U…` option.

Example

```
$ ./odb64luo -u xxx -p xxx -d traf -i D:TRAFODION.SEABASE.REGIONS

odb [2015-04-20 21:25:35]: starting ODBC connection(s)... 0
Connected to Trafodion

CREATE TABLE TRAFODION.SEABASE."REGIONS" ( REGION_ID INTEGER NOT NULL
, REGION_NAME VARCHAR(25)
);
```

The `&` wild card allows you to extract the DDL for multiple objects.

Example

The following command will extract the DDL for all tables in schema `tpch` starting with `P`:

```
$ ./odb64luo -u xxx -p xxx -d traf -i D:TRAFODION.TPCH.P%

odb [2015-04-20 21:33:43]: starting ODBC connection(s)... 0
Connected to Trafodion

CREATE TABLE TRAFODION.TPCH."PART" ( P_PARTKEY BIGINT NOT NULL
,P_NAME VARCHAR(55) NOT NULL
,P_MFGR CHAR(25) NOT NULL
,P_BRAND CHAR(10) NOT NULL
,P_TYPE VARCHAR(25) NOT NULL
,P_SIZE INTEGER NOT NULL
,P_CONTAINER CHAR(10) NOT NULL
,P_RETAILPRICE BIGINT NOT NULL
,P_COMMENT VARCHAR(23) NOT NULL
,PRIMARY KEY (P_PARTKEY)
);

CREATE TABLE TRAFODION.TPCH."PARTSUPP" ( PS_PARTKEY BIGINT NOT NULL
,PS_SUPPKEY BIGINT NOT NULL
,PS_AVAILQTY INTEGER NOT NULL
,PS_SUPPLYCOST BIGINT NOT NULL
,PS_COMMENT VARCHAR(199) NOT NULL
,PRIMARY KEY (PS_PARTKEY,PS_SUPPKEY)
);

odb [2015-04-20 21:33:45]: exiting. Session Elapsed time 2.069 seconds (00:00:02.069)
```

You should consider possible differences in text column length semantic when porting DDLs from one database to another; some databases use "character oriented" text columns length while others use a "byte oriented" semantic.

You can ask odb to multiply text column length when printing DDL using the switch `–U[non-wide_char_multiplier,wide_char_multiplier]`.

Example

```
$ ./odb64luo -u xxx -p xxx -d traf -i U2,4:TRAFODION.TPCH.P%

odb [2015-04-20 21:35:17]: starting ODBC connection(s)... 0
Connected to Trafodion

CREATE TABLE TRAFODION.TPCH."PART" ( P_PARTKEY BIGINT NOT NULL
,P_NAME VARCHAR(110) NOT NULL
,P_MFGR CHAR(50) NOT NULL
,P_BRAND CHAR(20) NOT NULL
,P_TYPE VARCHAR(50) NOT NULL
,P_SIZE INTEGER NOT NULL
,P_CONTAINER CHAR(20) NOT NULL
,P_RETAILPRICE BIGINT NOT NULL
,P_COMMENT VARCHAR(46) NOT NULL
,PRIMARY KEY (P_PARTKEY)
);

CREATE TABLE TRAFODION.TPCH."PARTSUPP" ( PS_PARTKEY BIGINT NOT NULL
,PS_SUPPKEY BIGINT NOT NULL
,PS_AVAILQTY INTEGER NOT NULL
,PS_SUPPLYCOST BIGINT NOT NULL
,PS_COMMENT VARCHAR(398) NOT NULL
,PRIMARY KEY (PS_PARTKEY,PS_SUPPKEY)
);

odb [2015-04-20 21:35:18]: exiting. Session Elapsed time 1.620 seconds (00:00:01.620)
```

The command in the above example multiplies the length of "non-wide" text fields by 2 and the length of wide text fields by 4.

Chapter 5. Load, Extract, Copy

5.1. Load Files

You can load a data file using `-l` option.

Example

```
$ odb64luo –u user –p xx –d dsn –l src=customer.tbl:tgt=TRAFODION.MAURIZIO.CUSTOMER \
:fs=\|:rows=1000:loadcmd=UL:truncate:parallel=4
```

This command:

- Loads the file named `customer.tbl` (`src=customer.tbl`)
- in the table `TRAFODION.MAURIZIO.CUSTOMER` (`tgt=TRAFODION.MAURIZIO.CUSTOMER`)
- using `|` (vertical bar) as a field separator (`fs=\|`)
- using `1000 rows` as row-set buffer (`rows=1000`)
- using `UPSERT USING LOAD` syntax to achieve better throughput as described in Trafodion Load and Transform Guide
- truncating the target table before loading (`truncate`)
- using `4 parallel threads` to load the target table (`parallel=4`)

5.1.1. Data Loading Operators

```
-l src=[-]file:tgt=table[:map=mapfile][:fs=fieldsep][:rs=recsep][:soe]
   [:skip=linestoskip][:ns=nullstring][:ec=eschar][:sq=stringqualifier]
   [:pc=padchar][:em=embedchar][:errmax=#max_err][:commit=auto|end|#rows|x#rs]
   [:rows=#rowset][:norb][:full][:max=#max_rec][:truncate][:show][:bpc=#][:bpwc=#]
   [:nomark][:parallel=number][:iobuff=#size][:buffsz=#size]][:fieldtrunc=\{0-4}]
   [:pre=\{@sqlfile}|\{[sqlcmd]}][:post=\{@sqlfile}|\{[sqlcmd]}][:ifempty]
   [:direct][:bad=[+]badfile][:tpar=#tables][:maxlen=#bytes][:time]
   [:xmltag=[+]element][:xmlord][:xmldump][:loadcmd=IN|UP|UL]
```

The following table describes each data loading operator:

Load option	Meaning
`src=<file>`	Input file. You can use the following keywords for this field: - `%t` expand to the (lower case) table name - `%T` expand to the (upper case) table name - `%s/%S` expand to the schema name - `%c/%C` expand to the catalog name - `stdin` load reading from the standard input - `-<file>` to load all files listed in `<file>` - `[hdfs][@host,port[,user]].<hdfspath>` to load files from Hadoop File System (via `libhdfs.so`) - `[mapr][@host,port[,user]].<maprpath>` to load files from MapR File System (via `libMapRClient.so`)
`tgt=<CAT.SCH.TAB>`	This is the target table
`fs=<char>|<code>`	This is the field separator. You can define the field separator: - as normal character (for example `fs=,`) - as ASCII decimal (for example `fs=44` — `44` means comma) - as ASCII octal value (for example `fs=054` — `054` means comma) - as ASCII hex value (for example `fs=x2C` — `x2C` means comma) Default field separator is `,` (comma)
`rs=<char>|<code>`	This is the record separator. You can define the record separator the same way as the field separator. Default record separator is `\n` (new line)
`pc=<char|code>`	Pad character used when loading fixed format files. You can use the same notation as the field separator.
`map=<mapfile>`	Uses mapfile to map source file to target table columns. See Map Source File Fields to Target Table Columns.
`skip=num`	Skips a given number of lines when loading. This can be useful to skip headers in the source file.
`max=num`	The max number of records to load. Default is to load all records in the input file
`ns=<nullstring>`	odb inserts NULL when it finds nullstring in the input file. By default the nullstring is the empty string
`sq=<char>|<code>`	The string qualifier character used to enclose strings. You can define the escape character the same way as the field separator.
`ec=<char>|<code>`	The character used as escape character. You can define the escape character the same way as the field separator. Default is `\` (back slash).
`rows=<num>|k<num>|m<num>`	This defines the size of the I/O buffer for each loading thread. You can define the size of this buffer in two different ways: 1. number of rows (for example: `rows=100` means 100 rows as IO buffer) 2.* buffer size in KB or MB (for example: `rows=k512` (512 KB buffer) or `rows=m20` (20MB buffer)) Default value is `100`.

Load option	Meaning
`bad=[+]file`	Where to write rejected rows. If you omit this parameter, then rejected rows is printed to standard error together with the error returned by the ODBC Driver. If you add a + sign in front of the file-name, odb **appends** to `<file>`instead of **create** the `<file>`.
`truncate`	Truncates the target table before loading.
`ifempty`	Loads the target table only if it contains no records.
`norb`	Loads `WITH NO ROLLBACK`.
`nomark`	Don't print the number of records loaded so far during loads.
`soe`	Stop On Error — stop as soon as odb encounters an error.
`parallel=num`	Number of loading threads. odb uses: - one thread to read from the input file and - as many threads as the parallel argument to write via ODBC. This option is database independent.
`errmax=num`	odb prints up to num error messages per rowset. Normally used with soe to limit the number of error messages printed to the standard error stream.
`commit=auto|end|#rows|x#rs`	Defines how odb commits the inserts. You have the following choices: - `auto` (default): Commit every single insert (see also rows load operator). - `end`: Commit when all rows (assigned to a given thread) have been inserted. - `#rows`: Commit every `#rows` inserted rows. - `x#rs`: Commit every `#rs` rowset (see `rows`)
`direct`	Adds `/*+ DIRECT */` hint to the insert statement. To be used with Vertica databases in order to store inserted rows **directly** into the Read-Only Storage (ROS). See Vertica's documentation.
`fieldtrunc={0-4}`	Defines how odb manages fields longer than the destination target column: - `fieldtrunc=0` (default): Truncates input string, print a warning and load the truncated field if the target column is a text field. - `fieldtrunc=1`: Like `fieldtrunc=0` but no warning message is printed. - `fieldtrunc=2`: Prints an error message and does NOT load the row. - `fieldtrunc=3`: Like `fieldtrunc=0` but tries to load the field even if the target column is NOT a text field. - `fieldtrunc=4`: Like fieldtrunc=3 but no warnings are printed. WARNING: the last two options could bring to unwanted results. For example, an input string like `2001-10-2345` is loaded as a valid 2001-10-23 if the target field is a `DATE`.
`em=<char>|<code>`	Character used to embed binary files. See Load Default Values. You can define the embed character the same way as the field separator. No default value.
`pre={@sqlfile}|{[sqlcmd]}`	odb runs a **single instance** of either `sqlfile` script or `sqlcmd` (enclosed between square brackets) on the **target system** immediately before loading the target table. You can, for example, CREATE the target table before loading it. Target table is not loaded if SQL execution fails and `Stop On Error (soe)` is set.

Load option	Meaning
`post={@sqlfile}|{[sqlcmd]}`	odb runs a **single instance** of either `sqlfile` script or `sqlcmd` (enclosed between square brackets) on the **target system** immediately after the target table has been loaded. You can, for example, update database stats after loading a table.
`tpar=num`	odb loads `num` tables in parallel when `src` is a list of files to be loaded.
`show`	odb prints what would be loaded in each column but no data is actually loaded. This is useful if you want to see how the input file *fits* into the target tables, Normally used to analyze the first few rows of CSV files (use `:max`). This option forces: - `parallel` to `1`. - `rows` to `1`. - `ifempty` to `false`. - `truncate` to `false`.
`maxlen=#bytes`	odb limits the amount of memory allocated in the ODBC buffers for CHAR/VARCHAR fields to `#bytes`.
`time`	odb prints a **time line** (milliseconds from start) for each insert.
`bpc=#`	Bytes allocated in the ODBC buffer for each (non wide) CHAR/VARCHAR column length unit. (Default: 1)
`bwpc=#`	Bytes allocated in the ODBC buffer for each (wide) CHAR/VARCHAR column length unit. (Default: 4)
`Xmltag=[+]tag`	Input file is XML. Load all *XML nodes* under the one specified with this option. If a plus sign is specified, then odb loads node-attributes values.
`xmlord`	By default, odb *matches* target table columns with XML node or attributes using their names. If this option is specified, then odb loads the first node/attribute to the first column, the second node/attribute to the second column and so on without checking node/attribute names.
`xmldump`	odb does not load the XML file content. Instead, XML attribute/tage names are printed to standard output so you can check what is going to be loaded.
`loadcmd`	SQL operation to be used for load. (Default: `INSERT`). `UPSERT` and `UPSERT USING LOAD` are also available for Trafodion.

You can load multiple files using different `-l` options. By default odb creates as many threads (and ODBC connections) as the sum of parallel load threads. You can limit this number using `-T` option.

Example

```
$ odb64luo –u user –p xx –d dsn –T 5 \
-l src=./data/%t.tbl.gz:tgt=TRAFODION.MAURO.CUSTOMER:fs=\
|:rows=m2:truncate:norb:parallel=4 \
-l src=./data/%t.tbl.gz:tgt=TRAFODION.MAURO.ORDERS:fs=\
|:rows=1000:truncate:norb:parallel=4 \
-l src=./data/%t.tbl.gz:tgt=TRAFODION.MAURO.LINEITEM:fs=\
|:rows=m10:truncate:norb:parallel=4
```

The above command truncates and loads the CUSTOMER, ORDERS and LINEITEM tables. The input files have the same name as the target tables -– in lower case). Loads are distributed among available threads this way:

Load Order	Thread 0	Thread 1	Thread2	Thread3	Thread4
Third	Read `lineitem.tbl`	Load `TRAFODION.MAURO.LINEITEM`	Load `TRAFODION.MAURO.LINEITEM`	Load `TRAFODION.MAURO.LINEITEM`	Load `TRAFODION.MAURO.LINEITEM`
Second	Read `orders.tbl`	Load `TRAFODION.MAURO.ORDERS`	Load `TRAFODION.MAURO.ORDERS`	Load `TRAFODION.MAURO.ORDERS`	Load `TRAFODION.MAURO.ORDERS`
First	Read `customer.tbl`	Load `TRAFODION.MAURO.CUSTOMER`	Load `TRAFODION.MAURO.CUSTOMER`	Load `TRAFODION.MAURO.CUSTOMER`	Load `TRAFODION.MAURO.CUSTOMER`

If you want to load more than one table in parallel you should use a number of threads defined as: `(parallel + 1) * tables_to_load_in_parallel`

You can load gzipped files without any special option. odb automatically checks input files and decompress them on the fly when needed.

odb using one single loading thread (`parallel=1`) is faster than without parallel — if you do not specify parallel, odb uses one thread to both read from file and write into the target table:

`Read buffer #1>Write Buffer #1>Read Buffer #2>Write Buffer #2>Read Buffer #3>Write Buffer#3>…`

`parallel=1` defines that there is one thread to read from file and one thread to write:

- `Read buffer #1>Read Buffer #2>Read Buffer #3>…`
- `Write Buffer #1>Write Buffer #2>Write Buffer #3>…`

Reading from file is **normally** much faster than writing via ODBC so a single *reading thread* can serve different *loading threads*. One could ask: what the *right* number of loading threads is?

In order to define the right number of loading threads you should run a few test and monitor the *Wait Cycles* reported by odb. Wait Cycles represent the number of times the *reading thread* had to wait for one *loading thread* to become available.

- When you have high *Wait Cycles/Total Cycles" ratio…* it's better to increase the number of writers.
- When the *Wait Cycles/Total Cycles* is less than 5%, adding more loading threads is useless or counterproductive.

5.2. Map Source File Fields to Target Table Columns

odb, *by default*, assumes that input files contain as many fields as the target table columns, and that file fields and target table columns are in the same order. This means that the first field in the input file is loaded in the first table column, second input field goes to the second column and so on.

If this basic assumption is not true and you need more flexibility to *link* input fields to target table columns, then odb provides mapping/transformation capabilities though **mapfiles**. By specifying `map=<mapfile>` load option you can:

- Associate any input file field to any table column
- Skip input file fields
- Generate sequences
- Insert constants
- Transform dates/timestamps formats
- Extract substrings
- Replace input file strings. For example: insert `Maurizio Felici` when you read `MF`
- Generate random values
- … and much more

A generic *mapfile* contains:

- **Comments** (line starting with #)
- **Mappings** to link input file fields to the corresponding target table columns.

Mappings use the following syntax:

```
<colname>:<field>[:transformation operator]
```

Where:

- `<colname>` is the target table column name. (Case sensitive)
- `<field>` is one of the following:
- The ordinal position (*`starting from zero`*) of the input file field.

  First input field is `0` (zero), second input field is `1` and so on
- `CONST:<CONSTANT>` to load a constant value
- `SEQ:<START>` to generate/load a sequence starting from `<START>`
- `IRAND:<MIN>:<MAX>` to generate/load a random integer between `<MIN>` and `<MAX>`

- DRAND:<MIN_YEAR>:<MAX_YEAR> to generate/load a random date (YYYY-MM-DD) between <MIN_YEAR> and <MAX_YEAR>
- TMRAND: to generate/load a random time (hh:mm:ss) between 00:00:00 and 23:59:59
- TSRAND: to generate/load a random timestamp (YYYY-MM-DD hh:mm:ss) between midnight UTC –- 01 Jan 1970 and the current timestamp
- CRAND:<LENGTH> generates/loads a string of <LENGTH> characters randomly selected in the following ranges: a-z, A-Z, 0-9
- NRAND:<PREC>:<SCALE> generates/loads a random NUMERIC field with precision <PREC> and scale <SCALE>
- DSRAND:<file> selects and loads a random line from <file>
- TXTRAND:<MIN_LENGTH>:<MAX_LENGTH>:<file>: selects and loads a random portion of test from <file> with length between <MIN_LENGTH> and <MAX_LENGTH>
- LSTRAND:<VALUE1,VALUE2,…> selects and loads a random value from <VALUE1,VALUE2,…>
- EMRAND:<MIN_ULENGTH>:<MAX_ULENGTH>:<MIN_DLENGTH>:<MAX_DLENGTH>:<SUFFIX1,SUFFIX2,…> generates and loads a string made of local@domain.suffix where:
  - local is a string of random characters (a-z, A-Z, 0-9) with length between <MIN_ULENGTH> and <MAX_ULENGTH>
  - domain is a string of random characters (a-z, A-Z, 0-9) with length between <MIN_DLENGTH> and <MAX_DLENGTH>
  - suffix is a randomly selected suffix from <SUFFIX1,SUFFIX2,…>
- CDATE: to load the current date (YYYY-MM-DD)
- CTIME: to load the current time (hh:mm:ss)
- CTSTAMP: to load the current timestamp (YYYY-MM-SS hh:mm:ss)
- FIXED:<START>:<LENGTH> to load fixed format fields made of <LENGTH> characters starting at <START>.

<START> starts from zero.

- EMPTYASNULL: loads empty strings in the input file as NULLs (default is to load empty string as empty strings)
- EMPTYASCONST:<CONSTANT>: loads empty fields in the input file as <CONSTANT>
- NULL: inserts NULL
- :transformation operators (optional):
- SUBSTR:<START>:<END>. For example, if you have an input field containing Tib:student a transformation rule like SUBSTR:3:6`m then `Tib is loaded into the database.

- TSCONV:<FORMAT>. Converts timestamps from the input file format defined through <FORMAT> to YYYY-MM-DD HH:MM:SS before loading. The input format is defined through any combination of the following characters:

Char	Meaning
b	abbreviated month name
B	full month name
d	day of the month
H	hour (24 hour format)
m	month number
M	Minute
S	Second
y	year (four digits)
D#	#decimal digits
.	ignore a single char
_	ignore up to the next digit

- DCONV:<FORMAT>. Converts dates from the input file format defined through <FORMAT> to YYYY-MM-DD (see TSCONV operator).

  Example: DCONV:B.d.y converts August,24 1991 to 1991-08-24

- TCONV:<FORMAT>. Converts times from the input file format defined through <FORMAT> to HH:MM:SS (see TSCONV operator).

- REPLACE:<READ>:<WRITTEN>. Loads the string <WRITTEN> when the input file fields contains <READ>. If the input file string doesn't match <READ>, then it is loaded as is.

  See Use mapfiles to Ignore and/or Transform Fields When Loading

- TOUPPER. Converts the string read from the input file to uppercase before loading.

  Example: proGRAmMEr –> PROGRAMMER

- TOLOWER. Converts the string read from the input file to lowercase before loading.

  Example: proGRAmMEr –> programmer

- FIRSTUP. Converts the first character of the string read from the input file to uppercase and the remaining characters to lowercase before loading.

  Example: proGRAmMEr –> Programmer

- `TRANSLIT:<LIST OF CHARS>:<LIST OF CHARS>`. Lets you to delete or change any character with another.

  Examples

  - `WORK:7:translit:Gp:HP` loads the seventh input field into the target column named `WORK` and replaces all `G` with `H` and all `p` with `P`
  - `WORK:7:translit:Gp\r:HP\d` behaves like the previous example but also deletes all `carriage returns` (`\r`)
  - `CSUBSTR`. This operator is somehow similar to `SUBSTR` but instead of using fixed position to extract substrings will use delimiting characters. For example, suppose your input fields (comma is the field separator) are:

    ```
    ... other fields...,name_Maurizio.programmer,...other fields
    ... other fields...,_name_Lucia.housewife, ...other fields...
    ... other fields...,first_name_Giovanni.farmer,... other fields...
    ... other fields...,_Antonella,... other fields...
    ... other fields...,Martina,...other fields...
    ... other fields...,Marco.student, ...other fields...
    ```

    Using a transformation like: `NAME:4:CSUBSTR:95:46` (where `95` is the ASCII code for `_` and `46` is the ASCII code for `.`) results in loading the following values into the target (`NAME`) column:

    ```
    Maurizio
    Lucia
    Giovanni
    Antonella
    Martina
    Marco
    ```

- `COMP`. Transform a packed binary `COMP` into a target database number.

  For example: `hex 80 00 00 7b` is loaded as `-123`

- `COMP3:PRECISION:SCALE`. Transform a packed binary `COMP-3` format into a target database number.

  For example: `hex 12 34 56 78 90 12 34 56 78 9b` is loaded as `-1234567890123456.789`

- `ZONED:PRECISION:SCALE`. Transform a packed binary `ZONED` format into a target database number.

  For example: `hex 31 32 33 34 35 36` is loaded as `+.123456`

5.3. Use mapfiles to Ignore and/or Transform Fields When Loading

The following example explains mapfile usage to skip/transform or generate fields. Suppose you have a target table like this:

```
+------+---------------+----+-------+------------+
|COLUMN|TYPE           |NULL|DEFAULT|INDEX       |
+------+---------------+----+-------+------------+
|ID    |INTEGER SIGNED |NO  |       |mf_pkey 1 U |
|NAME  |CHAR(10)       |YES |       |            |
|AGE   |SMALLINT SIGNED|YES |       |            |
|BDATE |DATE           |YES |       |            |
+------+---------------+----+-------+------------+
```

And an input file like this:

---

uno,00,**51**,due,*Maurizio*,tre,07 Mar 1959, ignore,remaining, fields

uno,00,**46**,due,*Lucia*,tre,13 Oct 1964, ignore, this

uno,00,**34**,due,*Giovanni*,tre,30 Mar 1976

uno,00,**48**,due,*Antonella*,tre,24 Apr 1962 ***

- **Bold text** represents age.
- *Italics text* represents name.
- Underline text represents birth date.

You want to load the marked fields into the appropriate column, ***generate*** a unique key for ID and ignore the remaining fields. In addition: you need to ***convert date format*** and replace all occurrences of `Lucia` with `Lucy`.

The following map file accomplishes these goals:

```
$ cat test/load_map/ml1.map
# Map file to load TRAFODION.MFTEST.FRIENDS from friends.dat
ID:seq:1                  # Inserts into ID column a sequence starting from 1
NAME:4:REPLACE:Lucia:Lucy # Loads field #4 into NAME and replace all occurrences of
Lucia with Lucy
AGE:2                     # Loads field #2 (they start from zero) into AGE
BDATE:6:DCONV:d.b.y       # Loads field #6 into BDATE converting date format from dd
mmm yyyy
```

Load as follows:

```
$ odb64luo –u user –p xx –d dsn \
  -l src=friends.dat:tgt=TRAFODION.MFTEST.FRIENDS:map=ml1.map:fs=,
```

5.4. Use mapfiles to Load Fixed Format Files

Suppose you have a target table like this:

```
+------+---------------+----+-------+------------+
|COLUMN|TYPE           |NULL|DEFAULT|INDEX       |
+------+---------------+----+-------+------------+
|NAME  |CHAR(10)       |YES |       |            |
|JOB   |CHAR(10)       |YES |       |            |
|BDATE |DATE           |YES |       |            |
+------+---------------+----+-------+------------+
```

And an input file like this:

```
GiovanniXXX30 Mar 1976YFarmer
Lucia   XXX13 Oct 1964YHousewife
Martina XXX28 Oct 1991Y?
Marco   XXX06 Nov 1994Y?
MaurizioXXX07 Mar 1959YProgrammer
```

You want to load the fixed-position fields into the appropriate columns and to ***convert date format***. Null values in the input file are represented by question marks. In this case you can use a mapfile like this:

```
~/Devel/odb $ cat test/fixed/ff.map
NAME:FIXED:0:8               <- insert into NAME characters starting at position 0,
length 8
BDATE:FIXED:11:11:DCONV:d.b.y <- insert into BDATE characters starting at col 11,
length 11 and convert date
JOB:FIXED:23:10              <- insert into JOB characters starting at position 23,
length 10
```

Load as follows:

```
$ odb64luo –u user –p xx –d dsn \
-l src=frends1.dat:tgt=TRAFODION.MFTEST.FRIENDS1:map=ff.map:ns=\?:pc=32
```

Where: `pc=32` identify the pad character in the input file (`space` = ASCII 32) and `ns=?` defines the null string in the input file.

5.5. Generate and Load Data

odb can generate and load data for testing purposes. The following example illustrates the odb capabilities in this area through an example.

Suppose you want to fill with test data a table like this:

```
CREATE TABLE TRAFODION.MAURIZIO."PERSON"
( PID BIGINT SIGNED NOT NULL
, FNAME CHAR(20) NOT NULL
, LNAME CHAR(20) NOT NULL
, COUNTRY VARCHAR(40) NOT NULL
, CITY VARCHAR(40) NOT NULL
, BDATE DATE NOT NULL
, SEX CHAR(1) NOT NULL
, EMAIL VARCHAR(40) NOT NULL
, SALARY NUMERIC(9,2) NOT NULL
, EMPL VARCHAR(40) NOT NULL
, NOTES VARCHAR(80)
, LOADTS TIMESTAMP(0)
, PRIMARY KEY (PID)
)
;
```

You can use a mapfile like this:

```
~/Devel/odb $ cat person.map
PID:SEQ:100
FNAME:DSRAND:datasets/first_names.txt
LNAME:DSRAND:datasets/last_names.txt
COUNTRY:DSRAND:datasets/countries.txt
CITY:DSRAND:datasets/cities.txt
BDATE:DRAND:1800:2012
SEX:LSTRAND:M,F,U
EMAIL:EMRAND:3:12:5:8:com,edu,org,net
SALARY:NRAND:9:2
EMPL:DSRAND:datasets/fortune500.txt
NOTES:TXTRAND:20:80:datasets/lorem_ipsum.txt
LOADTS:CTSTAMP
```

Where:

- `PID:SEQ:100` — Loads a sequence starting from `100` into `PID`
- `FNAME:DSRAND:datasets/first_names.txt` — Loads `FNAME` with a randomly selected value from `first_names.txt`. There are plenty of sample datasets available to generate all sort of data using *realistic* values.
- `LNAME:DSRAND:datasets/last_names.txt` — Loads `LNAME` with a random value from `last_names.txt`.
- `COUNTRY:DSRAND:datasets/countries.txt` — Loads `COUNTRY` with a random value from `countries.txt`.
- `CITY:DSRAND:datasets/cities.txt` — Loads `CITY` with a random value from `cities.txt`.
- `BDATE:DRAND:1800:2012` — Generates and loads into `BDATE` a random date between `1800-01-01` and `2012-12-31`.
- `SEX:LSTRAND:M,F,U` — Loads `SEX` with a random value in the `M`, `F`, `U` range.
- `EMAIL:EMRAND:3:12:5:8:com,edu,org,net` — Generates and loads a `local@domain.suffix email` addresses where:
- `local` is made of 3 to 12 random characters.
- `domain` is made of 5 to 8 random characters.
- `suffix` is `com`, `ord`, `edu`, or `net`.
- `SALARY:NRAND:9:2` — Generate and loads a random NUMERIC(9,2).
- `EMPL:DSRAND:datasets/fortune500.txt` — Loads `EMPL` with a random value from `fortune500.txt`.
- `NOTES:TXTRAND:20:80:datasets/lorem_ipsum.txt` — Loads `NOTES` with a random section of `lorem_ipsum.txt` with length between 20 and 80 characters`

- `LOADTS:CTSTAMP` — Loads the current timestamp into `LOADTS`.

You generate and load test data with a command like this:

```
$ bin/odb64luo -u user -p password -d traf -l src=nofile:
tgt=traf.maurizio.person:max=1000000:
map=person.map:rows=5000:parallel=8:loadcmd=INSERT
```

Please note `src=nofile" (it means *there is no input file*) and `max=1000000` (generate and load one million rows). The above command has generated and loaded 1M rows of *realistic* data in about ten seconds:

```
[0] odb Loading statistics:
[0] Target table: TRAFODION.MAURIZIO.PERSON
[0] Source: nofile
[0] Pre-loading time: 2.911 s
[0] Loading time: 7.466 s
[0] Total records read: 1,000,000
[0] Total records inserted: 1,000,000
[0] Total number of columns: 12
[0] Total bytes read: 3,963
[0] Average input row size: 0.0 B
[0] ODBC row size: 323 B (data) + 88 B (len ind) [0] Rowset size: 5,000
[0] Rowset buffer size: 2,006.83 KiB
[0] Load Performances (real data): 0.518 KiB/s
[0] Load Performances(ODBC): 42,243.161 KiB/s
[0] Reader Total/Wait Cycles: 200/16
```

5.6. Load Default Values

The simpler way to load database generated defaults is to ignore the associated columns in the map file. For example, suppose you have a table like this under Trafodion:

```
create table TRAFODION.maurizio.dtest
( id largeint generated by default as identity not null
, fname char(10)
, lname char(10) default 'Felici'
, bdate date
, comment varchar(100)
)
;
```

If you have an input file containing:

---

ignoreme,**Maurizio**,xyz,*commentM*, ignore,remaining, fields ignoreme,**Lucia**,xyz,*commentL*, ignore, this ignoreme,**Giovanni**,xyz,*commentG*, ignoreme,Antonella,xyz,commentA *****

- **Bold text** represents `fname`.
- *Italic text* represents `comment`.

and a map-file like this:

```
FNAME:1 BDATE:CDATE COMMENT:4
```

Then:

- First column (`ID`) is loaded with its default value (not in the map file)
- Second column (`FNAME`) is loaded with the second input field from file (`FNAME:1`)
- Third column (`LNAME`) is loaded with its default value (not in the map file)
- Fourth column (`BDATE`) is loaded with the Current Data generated by odb (`BDATE:CDATE`)
- Fifth column (`COMMENT`) is loaded with the fifth column in the input file (`COMMENT:4`)

5.7. Load Binary Files

Assuming that your back-end database (and your ODBC Driver) supports BLOB data types, or equivalent, you can use odb to directly load binary (or any other) files into a database column using the `[:em=char]` symbol to identify the file to

be loaded into that specific database field.

Example

Suppose you have a table like this (MySQL):

```
create table pers.myphotos
( id integer
, image mediumblob
, phts timestamp
)
;
```

Then, you can load a file like this:

```
$ cat myphotos.csv

001,@/home/mauro/images/image1.jpg,2012-10-21 07:31:21
002,@/home/mauro/images/image2.jpg,2012-10-21 07:31:21
003,@/home/mauro/images/image3.jpg,2012-10-21 07:31:21
```

by running a command like this:

```
$ odb64luo –u user –p xx –d dsn -l src=myphotos.csv:tgt=pers.myphotos:em=\@
```

odb considers the string following the “em” character as the path of the file to be loaded in that specific field.

odb does not load rows where the size of the input file is greater than the target database column.

5.8. Load XML Files

Trafodion odb supports loading XML files into tables, the key construct for XML files can be an element or an attribute.

5.8.1. Load XML Files Where Data is Stored in Element Nodes

1. Create a table.

```
./odb64luo -x "create table testxmlload(id int, name char(20))"
```

2. Suppose you have a xml file where data is stored in element nodes like the following.

```
-bash-4.1$ cat test.xml
<?xml version="1.0" encoding="UTF-8"?>
<data>
<id>1</id>
<name>hello</name>
</data>
```

To check what will be loaded before loading XML file into table, run the following command

`./odb64luo -l src=test.xml :tgt=testxmlload:xmltag=data:xmldump`

3. Load the test.xml file into the table, run the following command.

```
./odb64luo -l  src=test.xml:tgt=testxmlload:xmltag=data
```

`xmltag=data` means odb will load data from the element nodes. For more information, see Data Loading Operators.

5.8.2. Load XML Files Where Data is Stored in Attribute Nodes

1. Create a table.

```
./odb64luo -x "create table testxmlload(id int, name char(20))"
```

2. Suppose you have a XML file where data is stored in attribute nodes like the following.

```
-bash-4.1$ cat test.xml
<?xml version="1.0" encoding="UTF-8"?>
<data id="1" name="hello"></data>
```

To check what will be loaded before loading XML file into table, run the following command.

`./odb64luo -l src=test.xml:tgt=testxmlload:xmltag=data:xmldump`

3. Load the test.xml file into the table, run the following command.

```
./odb64luo -l src=test.xml:tgt=testxmlload:xmltag=+data
```

`xmltag=+data` (with a plus sign specified) means odb will load data from the attribute nodes. For more information, see Data Loading Operators.

5.9. Reduce the ODBC Buffer Size

odb allocates memory for the ODBC buffers during load/extract operations based on the max possible length of the source/target columns.

If you have a column defined as `VARCHAR(2000)`, then odb allocates enough space for 2,000 characters in the ODBC buffer.

If you know in advance that you never will load/extract 2,000 characters, then you can limit the amount of space allocated by odb. This reduces memory usage and increase performances because of the reduced network traffic.

Given the following table:

```
~/Devel/odb $ ./odb64luo -u xxx -p xxx -d traf -i D:TRAFODION.USR.TMX
odb [2015-04-20 21:41:38]: starting ODBCconnection(s)... 0
Connected to Trafodion
CREATE TABLE TRAFODION.USR."TMX"
( ID INTEGER NOT NULL
, NAME VARCHAR(400)
, PRIMARY KEY (ID)
)
;
```

And an input file that contains:

```
~/Devel/odb $ cat tmx.dat

1,Maurizio
2,Lucia
3,Martina
4,Giovanni
5,Marco
6,Roland
7,Randy
8,Paul
9,Josef
10,Some other name
```

The max length of the second field in this file is:

```
~/Devel/odb $ awk -F\, 'BEGIN\{max=0} \{if(NF==2)\{len=length($i);if(len>max)max=len}}
END\{print max}' tmx.dat
15
```

In this case you can use `:maxlen=15` to limit the amount of the ODBC buffer:

```
~/Devel/odb $ ./odb64luo -u xxx -p xxx -d traf -l
src=tmx.dat:tgt=usr.tmx:truncate:maxlen=15

odb [2015-04-20 21:46:11]:starting ODBC connection(s)... 0
Connected to Trafodion
[0.0.0]--- 0 row(s) deleted in 0.052s (prep 0.012s, exec 0.040s, fetch 0.000s/0.000s)
[0] 10 records inserted [commit]
[0] odb version 1.3.0 Load(2) statistics:
       [0] Target table: (null).USR.TMX
       [0] Source: tmx.dat
       [0] Pre-loading time: 1.254 s (00:00:01.254)
       [0] Loading time: 0.026 s(00:00:00.026)
       [0] Total records read: 10
       [0] Total records inserted: 10
       [0] Total number of columns: 2
       [0] Total bytes read: 99
       [0] Average input row size: 9.9 B
       [0] ODBC row size: *26 B (data) + 16 B (len ind)*
       [0] Rowset size: 100
       [0] Rowset buffer size: *4.10 KiB*
       [0] Load throughput (real data): 3.718 KiB/s
       [0] Load throughput (ODBC): 9.766 KiB/s
odb [2015-04-20 21:46:12]: exiting. Session Elapsed time 1.294 seconds (00:00:01.294)
```

If you do not specify this parameter odb allocates the buffer for the max possible length of each field:

```
~/Devel/odb $ ./odb64luo -u xxx -p xxx -d traf -l src=tmx.dat:tgt=usr.tmx:truncate

odb [2015-04-20 21:47:13]: starting ODBC connection(s)... 0
Connected to Trafodion
[0.0.0]--- 10 row(s) deleted in 0.107s (prep 0.012s, exec 0.095s, fetch 0.000s/0.000s)
[0] 10 records inserted [commit]
[0] odb version 1.3.0 Load(2) statistics:
       [0] Target table: (null).USR.TMX
       [0] Source: tmx.dat
       [0] Pre-loading time: 1.330 s (00:00:01.330)
       [0] Loading time: 0.032 s(00:00:00.032)
       [0] Total records read: 10
       [0] Total records inserted: 10
       [0] Total number of columns: 2
       [0] Total bytes read: 99
       [0] Average input row size: 9.9 B
       [0] ODBC row size: 411 B (data) + 16 B (len ind)
       [0] Rowset size: 100
       [0] Rowset buffer size: 41.70 KiB
       [0] Load throughput (real data): 3.021 KiB/s
       [0] Load throughput (ODBC): 125.427 KiB/s
odb [2015-04-20 21:47:14]: exiting. Session Elapsed time 1.373 seconds (00:00:01.373)
```

5.10. Extract Tables

You can use odb to extract tables from a database and write them to standard files (or named pipes).

Example

```
$ odb64luo –u user –p xx –d dsn –T 3 \
-e src=TRAFODION.MAURIZIO.LIN%:tgt=$\{DATA}/ext_%t.csv.gz:rows=m10:fs=\|:trim:gzip: \
-e src=TRAFODION.MAURIZIO.REGION:tgt=$\{DATA}/ext_%t.csv.gz:rows=m10:fs=\|:trim:gzip \
-e src=TRAFODION.MAURIZIO.NATION:tgt=$\{DATA}/ext_%t.csv.gz:rows=m10:fs=\|:trim:gzip
```

The example above:

- Extracts tables `REGION`, `NATION`, and all tables starting with `LIN` from `TRAFODION.MAURIZIO` schema.
- Saves data into files `ext_%t.csv.gz` (`%t` is expanded to the real table name).
- Compresses the output file (gzip) on the fly (uncompressed data never lands to disk).
- Trims text fields.
- Uses a 10 MB IO buffer.
- Uses three threads (ODBC connection) for the extraction process.

5.10.1. Extraction Options

```
-e {src={table|-file}|sql=<customsql>}:tgt=[+]file[:pwhere=where_cond]
   [:fs=fieldsep][:rs=recsep][:sq=stringqualifier][:ec=escape_char][:soe]
   [:ns=nullstring][es=emptystring][:rows=#rowset][:nomark][:binary][:fwc]
   [:max=#max_rec][:trim=[cCvVdt]][:rtrim][:cast][:multi][:efs=string]
   [:parallel=number][:gzip][:gzpar=wb??][:uncommitted][:splitby=column]

[:pre={@sqlfile}|{[sqlcmd]}[:mpre=\{@sqlfile}|{[sqlcmd]}[:post={@sqlfile}|{[sqlcmd]}]
   [tpar=#tables][:time][:nts][:cols=[-]columns]][:maxlen=#bytes][:xml]
```

The following table describes each extract operator:

Extract option	Meaning
`src=<CAT.SCH.TAB>|-file`	Defines the source table(s). You can use: - a single table name (for example TRAFODION.MFTEST.LINEITEM) - a group of tables (for example TRAFODION.MFTEST.LIN%) - a file containing a list of tables to extract (`-` should precede the filename)
`sql=<sql>`	A custom SQL command you can use to extract data. This is **alternative** to `src=`.
`tgt=[+]file`	Output file. You can use the following keywords for this field: - `%t/%T` expands to the (lower/upper case) table name - `%s/%S` expands to the (lower/upper case) schema name - `%c/%C` expands to the (lower/upper case) catalog name - `%d` expands to the extraction date (YYYYMMDD format) - `%D` expands to the extraction date (YYYY-MM-DD format) - `%m` expands to the extraction time (hhmmss format) - `%M` expands to the extraction time (hh:mm:ss format) - `stdout` prints the extracted records to the standard output. If you add a + sign in front of the file-name, then odb **appends** to `file` instead of **creates** `file`. `hdfs./<hdfspath>/<file>` to write exported table under the Hadoop File Distributed System (HDFS).
`fs=<char>|<code>`	Field separator. You can define the field separator as: - a normal character (for example `fs=,`) - ASCII decimal (for example `fs=44` - 44 means comma) - ASCII octal value (for example `fs=054` – 054 means comma) - ASCII hex value (for example `fs=x2C` – x2C means comma) The default field separator is `,` (comma)
`rs=<char>|<code>`	Record separator. You can define the record separator the same way as the field separator. The default record separator is `\n` (new line)
`max=num`	Max number of records to extract. The default is to extract all records
`sq=<char>|<code>`	The string qualifier character used to enclose strings. You can define the string qualifier the same way as the field separator
`ec=<char>|<code>`	Character used as escape character. You can define the escape character the same way as the field separator. Default is `\` (back slash).
`rows=<num>|k<num>|m<num>`	Defines the size of the I/O buffer for each extraction thread. You can define the size of this buffer in two different ways: - number of rows (for example: `rows=100` means 100 rows as IO buffer) - buffer size in kB or MB (for example: `rows=k512` (512 kB buffer) or `rows=m20` (20MB buffer))

Extract option	Meaning
`ns=<nullstring>`	How odb represents NULL values in the output file. Default is the empty string (two field separators one after the other)
`es=<emptystring>`	How odb represents VARCHAR empty strings (NOT NULL with zero length) values in the output file. Default is the empty string (two field separators one after the other)
`gzpar=<params>`	This are extra parameters you can pass to *tune* the gzip compression algorithm. **Examples** - `gzpar=wb9`: max compression (slower) - `gzpar=wb1`: basic compression (faster) - `gzpar=wb6h`: Huffman compression only - `gzpar=wb6R`: Run-length encoding only
`trim[=<params>]`	Accept the following optional parameters: - `c` trims[1] from CHAR[2]. - `C` trims trailing spaces from CHAR[2] - `v` trims leading spaces from VARCHAR fields - `V` trims trailing spaces from VARCHAR fields - `d` trims trailing zeros after decimal sign. Example: `12.3000` is extracted as `12.3`. - `t` trims decimal portion from TIME/TIMESTAMP fields. For example: `1999-12-19 12:00:21.345` is extracted as `1999-12-19 12:00:21`. **Trim Examples** `:trim=cC` —> *trims leading/trailing spaces from CHAR fields*. `:trim=cCd` —> *trims leading/trailing spaces from CHARs and trailing decimal zeroes*. If you do not specify any argument for this operator odb uses `cCvV`. In other words `:trim:` is a shortcut for `:trim=cCvV:`.
`nomark`	Don’t print the number of records extracted so far by each thread.
`soe`	Stop On Error. odb stop as soon as it encounters an error.
`parallel=num`	odb uses as many threads as the parallel argument to extract data from partitioned source tables. **You have to use splitby.** Each thread takes care of a specific range of the source table partitions. For example if you specify `parallel=4` and the source table is made of 32 partitions, then odb starts **four** threads (four ODBC connections): - thread 0 extracts partitions 0-7 - thread 1 extracts partitions 8-15 - thread 2 extracts partitions 16-23 - thread 3 extracts partitions 24-31

Extract option	Meaning
`multi`	This option can be used in conjunction with parallel operator to write as many output files as the number of extraction threads. Output file names are built adding four digits at the end of the file identified by the `tgt` operator. For example, with `src=trafodion.mauro.orders:tgt=%t.csv:parallel=4:multi` odb writes into the following output files: - `orders.csv.0001` - `orders.csv.0002` - `orders.csv.0003` - `orders.csv.0004`
`pwhere=<where condition>`	This option is used in conjunction with parallel limiting the extraction to records satisfying the where condition. NOTE: The where condition is limited to columns in the source table. For example: you want to extract records with `TRANS_TS > 1999-12-12 09:00:00` from the source table TRAFODION.MAURO.MFORDERS using eight parallel streams to a single, gzipped, file having the same name as the source table: `src=trafodion.mauro.mforders:tgt=%t.gz:gzip:parallel=8:pwhere=[TRANS_TS > TIMESTAMP '1999-12-12 09:00:00']…` You can enclose the where condition between square brackets to avoid a misinterpretation of the characters in the where condition.
`errmax=num`	odb prints up to num error messages per rowset. Normally used with soe to limit the number of error messages printed to the standard error stream.
`uncommitted`	Adds FOR READ UNCOMMITTED ACCESS to the select(s) command(s).
`rtrim`	RTRIM() CHAR columns on the server. From a functional point of view this is equivalent to `trim=C` but `rtrim` is executed on the server so it saves both client CPU cycles and network bandwidth.
`cast`	Performs a (server side) cast to VARCHAR for all non-text columns. Main scope of this operator is to *move* CPU cycles from the client to the database server. It increases network traffic. To be used when: - the extraction process is CPU bound on the client AND - network has a lot of available bandwidth AND - database server CPUs are not *under pressure*. Tests extracting a table full of NUMERIC(18,4), INT and DATES shows: - client CPU cycles down ~50% on the client - network traffic up ~40%

Extract option	Meaning

`splitby=<column>`	This operator let you to use parallel extract from any database. **<column> has to be a SINGLE, numeric column**. odb calculates min()/max() value for `<column>` and assign to each <parallel> thread the extraction of the rows in its *bucket*.
	
	For example, if you have:
	
	`…:splitby=emp_id:parallel=4…`
	
	with `min(emp_id)=1` and `max(emp_id)=1000`, the four threads extract the following rows:
	
	`thread #0 emp_id >=1 and emp_id < 251`
	`thread #1 emp_id >=251 and emp_id < 501`
	`thread #2 emp_id >=501 and emp_id < 751`
	`thread #3 emp_id >=751 and emp_id < 1001` (odb uses max(emp_id) + 1)
	
	If the values are not equally distributed, then data extraction is de-skewed.

`pre={@sqlfile}|{[sqlcmd]`	odb runs a **single instance** of either the `sqlfile` script or `sqlcmd` SQL command (enclosed between square brackets) on the **source system** immediately before table extraction. Source table won't be extracted if SQL execution fails and Stop On Error is set.

`mpre={@sqlfile}|{[sqlcmd]}`	Each odb thread runs either sqlfile script or sqlcmd SQL command (enclosed between square brackets)on the **source system** immediately before table extraction. You can use `mpre` to set database specific features **for each extraction thread**.
	
	Examples
	
	1. You want **Trafodion** to ignore missing stats warning. Then you can run via `mpre` a SQL script containing:
	
	`control query default HIST_MISSING_STATS_WARNING_LEVEL '0';`
	
	2. You want **Oracle** to extract dates in the `YYYY-MM-DD hh:mm:ss` format. Then you can run via `mpre` a script containing:
	
	`ALTER SESSION SET NLS_DATE_FORMAT='YYYY-MM-DD HH:MI:SS'`

`post={@sqlfile}|{[sqlcmd]}`	odb runs a **single instance** of either a `sqlfile` script or `sqlcmd` SQL command (enclosed between square brackets) on the **source system** immediately after table extraction.
`tpar=num`	odb extracts `num` tables in parallel when `src` is a list of files to be loaded.
`maxlen=#bytes`	odb limits the amount of memory allocated in the ODBC buffers for CHAR/VARCHAR fields to `#bytes`.
`xml`	Writes output file in XML format
`time`	odb prints a *timeline* (milliseconds from start).

1. The following characters are considered *spaces*: `blank`, `tab`, `new line`, `carriage return`, `form feed`, and `vertical tab`.
2. When the source table column is defined as NOT NULL and the specific field contains only blanks, odb leaves in the output file one single blank. This helps to distinguish between NULL fields (`<field_sep><field_sep>`) and NOT NULL fields containing all blanks (`<field_sep><blank><field_sep>`).

5.11. Extract a List of Tables

You can use odb to extract all tables listed in a file.

Example

```
~/Devel/odb $ cat tlist.txt

# List of tables to extract src=TRAFODION.MAURIZIO.ORDERS
src=TRAFODION.MAURIZIO.CUSTOMER src=TRAFODION.MAURIZIO.PART
src=TRAFODION.MAURIZIO.LINEITEM
```

You can extract all these tables by running:

```
$ odb64luo –u user –p xx –d dsn -e src=-tlist.txt:tgt=%t_%d%m:rows=m20:sq=\”
```

Please note the `src=-tlist.txt`.

5.12. Copy Tables From One Database to Another

odb can directly copy tables from one data-source to another. For example, from Trafodion to Teradata or vice-versa). Data ***never lands to disk*** when using this option.

The target table has to be be created in advance and should have a compatible structure.

5.12.1. Copy Operators

```
-cp src={table|-file:tgt=schema[.table][pwhere=where_cond][:soe][:nts]
   [:truncate][:rows=#rowset][:nomark][:max=#max_rec][:fwc][:bpwc=#]
   [:parallel=number][errmax=#max_err][:commit=auto|end|#rows|x#rs][:time]
   [:direct][:uncommitted][:norb][:splitby=column][:pre={@sqlfile}|{[sqlcmd]}]
   [:post={@sqlfile}|{[sqlcmd]}][:mpre={@sqlfile}|{[sqlcmd]}][:ifempty]
   [:loaders=#loaders][:tpar=#tables][:cols=[-]columns]
   [sql={[sqlcmd]|@sqlfile|-file}[:bind=auto|char|cdef]
   [tmpre={@sqlfile}|{[sqlcmd]}][seq=field#[,start]]
```

Complete list of the Copy Operators:

Copy Operator	Meaning
`src=<CAT.SCH.TAB>|-file`	Defines the source table(s). You can use: - a single table (for example: TRAFODION.MFTEST.LINEITEM) - a group of tables (for example: TRAFODION.MFTEST.LIN%) - a file containing a list of tables to copy ('-' should precede the filename)
`tgt=<CAT.SCH.TAB>`	Target table(s). You can use the following keywords for this field: - `%t/%T`: Expands to the (lower/upper case) source table name. - `%s/%S`: Expands to the (lower/upper case) source schema name. - `%c/%C`: Expands to the (lower/upper case) source catalog name.
`sql={[sqlcmd]|@sqlfile|-file}`	odb uses a generic SQL — instead of a *real* table — as source.
`max=num`	This is the max number of records to copy. Default is to copy all records in the source table.
`rows=<num>|k<num>|m<num>`	Defines the size of the I/O buffer for each copy thread. You can define the size of this buffer in two different ways: - number of rows (for example: `rows=100` means 100 rows as IO buffer) - buffer size in kB or MB (for example: `rows=k512` (512 kB buffer) or `rows=m20` (20MB buffer))
`truncate`	Truncates the target table before loading.
`ifempty`	Loads the target table only if empty.
`nomark`	Don't print the number of records loaded so far during loads.
`soe`	Stop On Error. odb stops as soon as it encounters an error.

Copy Operator	Meaning
`parallel=num`	odb uses two kinds of threads: • To extract data from partitioned source table. The number of the threads is as many as the parallel argument. • To write data to the target table. The number of the threads is equal to **parallel argument** * **number of loaders**. **Example** If you specify parallel argument = 4 and the source table has 32 partitions, then odb start: • 4 threads (4 ODBC connections) to read from the source table. • 8 threads (8 ODBC connections = 4 [parallel argument] * 2 [default number of loaders]) to write to the target table. [cols="1,2a"] !=== ! Thread ! Task ! Thread 0 ! Extracts partitions 0-7 from source. ! Thread 1 and Thread 2 ! Write data extracted from thread 0 to target. ! Thread 3 ! Extracts partitions 8-15 from source. ! Thread 4 and Thread 5 ! Write data extracted from thread 3 to target. ! Thread 6 ! Extracts partitions 16-23 from source. ! Thread 7 and Thread 8 ! Write data extracted from thread 6 to target. ! Thread 9 ! Extracts partitions 24-31 from source. ! Thread 10 and Thread 11 ! Write data extracted from thread 9 to target. !=== **You have to specify splitby.**
`pwhere=<where condition>`	Used in conjunction with parallel to copy only records satisfying the where condition. **Note:** The where condition is limited to columns in the source table. **Example** You want to copy records with `TRANS_TS > 1999-12-12 09:00:00` from the source table TRAFODION.MAURO.MFORDERS using eight parallel streams to a target table having the same name as the source table: `src=trafodion.mauro.mforders:tgt=trafodion.dest_schema.%t:parallel=8:pwhere=[TRANS_TS > TIMESTAMP '1999-12-12 09:00:00']…` You can enclose the where condition between square brackets to avoid a misinterpretation of the characters in the where condition.
`commit=auto|end|#rows|x#rs`	Defines how odb will commit the inserts. You have the following choices: - `auto` (Default) &8212; Commits every single insert (see also rows load operator). `end` commits when all rows (assigned to a given thread) have been inserted. - `#rows` — Commits every `#rows` copied rows. - `x#rs` — Commits every `#rs` rowset copied. (See `:rows`)
`direct`	Adds `/*+ DIRECT */` hint to the insert statement. To be used with Vertica databases in order to store inserted rows *directly* into the Read-Only Storage (ROS). See Vertica's documentation.
`errmax=num`	odb prints up to num error messages per rowset. Normally used with soe to limit the number of error messages printed to the standard error stream.
`uncommitted`	Adds `FOR READ UNCOMMITTED ACCESS` to the `select(s) command(s)`.

Copy Operator	Meaning
splitby=<column>	Lets you to use parallel copy from any database. **<column> has to be a SINGLE, numeric column**. odb calculates min()/max() value for <column> and assigns to each <parallel> thread the extraction of the rows in its *bucket*. For example, if you have: …:splitby=emp_id:parallel=4… with min(emp_id)=1 and max(emp_id)=1000, then the four threads extracts the following rows: thread #0 emp_id >=1 and emp_id < 251 thread #1 emp_id >=251 and emp_id < 501 thread #2 emp_id >=501 and emp_id < 751 thread #3 emp_id >=751 and emp_id < 1001 (odb uses max(emp_id) + 1) If the values are not equally distributed data extraction is de-skewed.
pre={@sqlfile}|{[sqlcmd]}	odb runs a **single instance** of either a sqlfile script or sqlcmd (enclosed between square brackets) on the **target system** immediately before loading the target table. You can, for example, CREATE the target table before loading it. The target table isn't loaded if SQL execution fails and Stop On Error is set.
mpre={@sqlfile}|{[sqlcmd]}	Each odb thread runs either a sqlfile script or sqlcmd (enclosed between square brackets) on the **source system** immediately before loading the target table. You can use mpre to set database specific features for each thread.
tmpre={@sqlfile}|{[sqlcm d]}	Each odb thread runs either a sqlfile script or sqlcmd (enclosed between square brackets) on the **target system** immediately before loading the target table. You can use mpre to set database specific features for each thread.
post={@sqlfile}|{[sqlcmd]}	odb runs a **single instance** of either a sqlfile script or sqlcmd (enclosed between square brackets) on the **target system** immediately after the target table has been loaded. You can, for example, update database stats after loading a table.
tpar=num	odb copies num tables in parallel when src is a list of files to be loaded.
loaders=num	odb uses num load threads for each extract thread. Default is two loaders per extractor,
fwc	Force Wide Characters. odb considers SQL_CHAR/SQL_VARCHAR fields as they were defined SQL_WCHAR/SQL_WVARCHAR.
bpwc=#	odb internally allocates 4 bytes/char for SQL_WCHAR/SQL_WVARCHAR columns. You can modify the number of bytes allocated for each char using this parameter.
bind=auto|char|cdef	odb can bind columns to ODBC buffer as characters (char) or C Default data types (cdef). The default (auto) uses cdef if SRC/TGT use the same database or char if SRC/TGT databases differ.
seq=field#[,start]	odb adds a sequence when loading the target system on column number field#. You can optionally define the sequence start value. (Default: 1)
time	odb prints a *timeline* (milliseconds from start).

When copying data from one data source to another, odb needs user/password/dsn for both source and target system.

User credentials and DSN for the target system are specified this way:

```
$ odb64luo –u src_user:tgt_user –p src_pwd:tgt:pwd –d src_dsn:tgt_dsn ... -cp
src=...:tgt=...
```

5.13. Copy a List of Tables

You can use odb to copy a list of tables from one database to another.

Example

```
~/Devel/odb $ cat tlist.txt

# List of tables to extract
src=TRAFODION.MAURIZIO.ORDERS
src=TRAFODION.MAURIZIO.CUSTOMER
src=TRAFODION.MAURIZIO.PART
src=TRAFODION.MAURIZIO.LINEITEM
```

You can extract all these tables by running:

```
$ odb64luo –u user1:user2 –p xx:yy –d dsn1:dsn2 \
-cp src=-tlist.txt:tgt=tpch.stg_%t:rows=m2:truncate:parallel=4 -T 8
```

Please note the `src=-tlist.txt`. This command copies:

Source	Target
`TRAFODION.MAURIZIO.ORDERS`	`tpch.stg_orders`
`TRAFODION.MAURIZIO.CUSTOMER`	`tpch.stg_customer`
`TRAFODION.MAURIZIO.PART`	`tpch.stg_part`
`TRAFODION.MAURIZIO.LINEITEM`	`tpch.stg_lineitem`

Optionally, you can define any other *command line* options in the input file.

Example

Using different *splitby columns*.

```
~/Devel/odb $ cat tlist2.txt

# List of tables to extract and their “splitby columns”
src=TRAFODION.MAURIZIO.ORDERS:splitby=O_ORDERKEY
src=TRAFODION.MAURIZIO.CUSTOMER:splitby=C_CUSTOMERKEY
src=TRAFODION.MAURIZIO.PART:splitby=P_PARTKEY
src=TRAFODION.MAURIZIO.LINEITEM:splitby=L_PARTKEY
```

5.14. Case-Sensitive Table and Column Names

Your database configuration determines whether you can use case sensitive table/column names. odb maintains table/column case sensitiveness when they are enclosed in double quotes.

Example

The following commands create a `TRAFODION.MAURIZIO.Names` table made of three columns: “name”, “NAME” and “Name”.

```
create table trafodion.maurizio."Names"
( "name" char(10)
, "NAME" char(10)
, "Name" char(10)
)
no partitions;
```

Double quotes have to be escaped under *nix. A few examples:

```
~/Devel/odb $ ./odb64luo -i T:trafodion.maurizio.\"Names\"
~/Devel/odb $ ./odb64luo -x "select  from trafodion.maurizio.\"Names\""
~/Devel/odb $ ./odb64luo -l src=names.txt:tgt=trafodion.maurizio.
\"Names\":map=names.map:pc=32
```

You can omit double quotes around column names when using *mapfiles*.

5.15. Determine Appropriate Number of Threads for Load/Extract/Copy/Diff

If you have to load/extract or copy multiple tables in parallel the best option is to use the options `:tpar=number` and `:parallel=number`. `:tpar` defines how many tables have to be copied/extracted in parallel; `:parallel` defines how many *data streams* to use for each table. This way, odb automatically allocates and start the “right” number of threads.

A rule of thumb when copying/loading or extracting tables is to use as many *data streams* as: `min(number of middle-tier CPUs, number of source CPUs, number of target CPUs)`

The number of threads started for each *data stream* depend on the operation type:

Operation	Total threads	Explanation	Example with `parallel=4`
Load	parallel + 1	One thread to read from file + one thread per parallel to load.	5
Extract	parallel	One thread per parallel to extract.	4
Copy	parallel * (1+loaders)	Two threads per parallel: read from source and write to target.	12 (if loaders=2)
Diff	parallel * 3	Three threads per parallel: read from source, read from target, compare.	12

5.16. Integrating With Hadoop

There are basically two ways to integrate a generic database with Hadoop using odb:

1. **Use HIVE (Hadoop DWH) and its ODBC Driver**: odb can access HIVE like any other *normal* relational database. For example, you can copy to from HIVE and other databases using odb’s copy option.
2. **Add the `hdfs.` prefix** to the input or output file during loads/extracts*: The file is read/written from/to Hadoop. odb interacts directly with the HDFS file system using **libhdfs**.

   This option is currently available only under Linux.

Chapter 6. Comparing Tables (Technology Preview)

You can use odb to compare two tables **with the same structure** on different databases. odb does the following to compare two tables:

1. Extracts source/target tables ordered by Primary Key or any other set of columns.
2. Compare source/target ODBC buffers without *unpacking* them into columns/rows.

Each *comparison stream* is made up of three threads:

- One thread reading from the source table.
- One thread reading from the target table.
- One thread comparing the source/target buffers.

These three threads work in parallel: the *compare* thread checks `buffer N` while the other two threads extract the net block of data from the source/target database in parallel.

You can have multiple *triplets* working in parallel on different section of the table using the `splitby` operator.

Example

```
$ odb64luo -u mailto:maurizio.felici@hp.com[MFELICI:maurizio.felici@hp.com] \
-d MFELICI:VMFELICI \
-p xx:yy –diff src=trafodion.maurizio.lineitem:tgt=mftest.lineitem:&#42; \
key=l_orderkey,l_linenumber:output=lineitem.diff:
rows=m2:print=IDC:&#42;splitby=l_orderkey&#42;:parallel=8
```

The command above compares two tables using eight streams (`parallel=8`) made of three threads each.

The comparison threads use double buffering and advanced memory-comparison techniques. odb can provide the following information in output as a CSV file:

- Missing rows on target (`D` – deleted – rows) based on the **key** columns.
- New rows on target (`I` – inserted – rows) based on the **key** columns.
- Changed rows (same **key** columns but with different values in other fields).

  For these rows odb can print the original source version (`C` rows) and/or the modified target version (`U` rows).

Example

odb output when comparing two tables:

```
$ cat lineitem.diff

DTYPE,L_ORDERKEY,L_LINENUMBER,L_SUPPKEY,L_PARTKEY,L_QUANTITY,L_EXTENDEDPRICE,L_DISCOUNT
,L_TAX,L_RETURNFLAG,
L_LINESTATUS,L_SHIPDATE,L_COMMITDATE,L_RECEIPTDATE,L_SHIPINSTRUCT,L_SHIPMODE,L_COMMENT
D,4532896,1,5974,100953,42.00,82065.90,0.03,0.00,R,F,1994-12-15,1995-01-17,1995-01-
07,COLLECT COD,TRUCK,leep across the ca
D,4532896,2,2327,102326,48.00,63759.36,0.07,0.05,A,F,1995-02-18,1994-12-10,1995-03-
12,TAKE BACK RETURN,RAIL,usly regular platelets. careful
D,4532896,3,612,193054,12.00,13764.60,0.05,0.02,R,F,1994-11-17,1994-11-23,1994-12-
06,COLLECT COD,SHIP,s haggle quickly. ideas after the
D,4532896,4,9867,47362,36.00,47136.96,0.10,0.06,A,F,1995-01-05,1994-11-29,1995-01-
06,COLLECT COD,RAIL,s haggle carefully bo
D,4532896,5,9576,2075,19.00,18564.33,0.00,0.05,R,F,1994-11-26,1995-01-17,1994-12-
03,COLLECT COD,TRUCK,en sauternes integrate blithely alon
D,4532896,6,1016,68509,9.00,13297.50,0.07,0.00,R,F,1995-02-16,1995-01-05,1995-02-
24,TAKE BACK RETURN,RAIL,ily above the blithel
C,1652227,3,2298,87281,28.00,35511.84,0.06,0.05,R,F,1993-05-04,1993-03-12,1993-05-
12,TAKE BACK RETURN,MAIL,lly final acco
U,1652227,3,2298,87281,99.99,35511.84,0.06,0.05,R,F,1993-05-04,1993-03-12,1993-05-
12,TAKE BACK RETURN,MAIL,lly final acco
D,3456226,1,8161,148160,22.00,26579.52,0.06,0.02,A,F,1994-06-26,1994-06-08,1994-07-
10,DELIVER IN PERSON,FOB,uriously. furio
D,3456226,2,6293,108762,20.00,35415.20,0.10,0.05,R,F,1994-05-07,1994-06-03,1994-05-
15,NONE,RAIL,ously bold requests along the b
```

```
D,3456226,3,4542,159511,33.00,51826.83,0.05,0.03,A,F,1994-07-04,1994-05-15,1994-07-
26,NONE,FOB,wake carefully al
D,3456226,4,154,95135,33.00,37294.29,0.04,0.08,A,F,1994-05-27,1994-05-10,1994-06-
14,DELIVER IN PERSON,AIR,ests. unusual dependencies wake fluffily
D,3456226,5,9027,126514,31.00,47755.81,0.08,0.01,R,F,1994-06-13,1994-06-18,1994-07-
10,TAKE BACK RETURN,FOB,according to the arefully regular instruct
D,3456226,6,8477,110943,14.00,27355.16,0.03,0.01,R,F,1994-07-03,1994-05-28,1994-07-
13,TAKE BACK RETURN,FOB,onic accounts. ironic,pend
D,3456226,7,1773,4272,34.00,39993.18,0.08,0.00,A,F,1994-05-01,1994-05-29,1994-05-
15,TAKE BACK RETURN,MAIL,ounts are finally ca
D,3456227,7,3722,101211,22.00,26668.62,0.02,0.01,N,O,1997-12-16,1998-02-05,1997-12-
19,NONE,TRUCK,uriously even platelets are fu
I,3456227,8,3722,101211,22.00,26668.62,0.02,0.01,N,O,1997-12-16,1998-02-05,1997-12-
19,NONE,TRUCK,uriously even platelets are fu
I,9999999,1,8161,148160,22.00,26579.52,0.06,0.02,A,F,1994-06-26,1994-06-08,1994-07-
10,DELIVER IN PERSON,FOB,uriously. furio
I,9999999,2,6293,108762,20.00,35415.20,0.10,0.05,R,F,1994-05-07,1994-06-03,1994-05-
15,NONE,RAIL,ously bold requests along the b
I,9999999,3,4542,159511,33.00,51826.83,0.05,0.03,A,F,1994-07-04,1994-05-15,1994-07-
26,NONE,FOB,wakecarefully al
I,9999999,4,154,95135,33.00,37294.29,0.04,0.08,A,F,1994-05-27,1994-05-10,1994-06-
14,DELIVER IN PERSON,AIR,ests. unusual dependencies wake fluffily
I,9999999,5,9027,126514,31.00,47755.81,0.08,0.01,R,F,1994-06-13,1994-06-18,1994-07-
10,TAKE BACK RETURN,FOB,according to the carefully regular instruct
I,9999999,6,8477,110943,14.00,27355.16,0.03,0.01,R,F,1994-07-03,1994-05-28,1994-07-
13,TAKE BACK RETURN,FOB,onic accounts. ironic, pend
I,9999999,7,1773,4272,34.00,39993.18,0.08,0.00,A,F,1994-05-01,1994-05-29,1994-05-
15,TAKE BACK RETURN,MAIL,ounts are finally ca
```

As you can see the first column defines the type of difference.

6.1. Diff Operators

```
-diff
src={table|-file}:tgt=table:[key=columns][:output=[+]file][:pwhere=where_cond]
[:pwhere=where_cond][:nomark][:rows=#rowset][:odad][:fs=fieldsep][:time]
[:rs=recsep][:quick][:splitby=column][:parallel=number][:max=#max_rec]
[:print=[I][D][C]][:ns=nullstring][:es=emptystring][:fwc][:uncommitted]
[:pre={@sqlfile}|{[sqlcmd]}][:post={@sqlfile}|{[sqlcmd]}][tpar=#tables]
```

Detailed Descriptions of the Copy Operators:

Diff Operator	Meaning
`src=<CAT.SCH.TAB>|-file`	Defines the source table(s). You can use: - A single table (for example: TRAFODION.MFTEST.LINEITEM) - A file containing a list of tables to compare (`-` should precede the filename)
`tgt=<CAT.SCH.TAB>`	Target table.

Diff Operator	Meaning
`key=column[,column,…]`	Define show to order records extracted from both source and target table. **If you do not specify any key column, then odb uses the Primary Key.**
`output=[+]file`	Output file where the differences are reported. You can use stdout to print odb output on the standard output. A + sign in front of the file-name tells odb to **append** to an existing file. Default value: `stdout`
`fs=<char>|<code>`	Field separator of the output file. You can define the field separator as follows: - Normal character (for example `fs=,`) - ASCII decimal (for example `fs=44` — 44 means comma) - ASCII octal value (for example `fs=054` — 054 means comma) - ASCII hex value (for example fs=x2C — x2C means comma) The default field separator is `,` (comma).
`rs=<char>|<code>`	Record separator used in the output file. You can define the record separator the same way as the field separator. The default record separator is `\n` (new line).
`max=num`	The max number of records to compare. Default is to compare all records.
`rows=<num>|k<num>|m<num>`	Defines the size of the I/O buffer for each extraction thread. You can define the size of this buffer in two different ways: - number of rows (for example: `rows=100` means 100 rows as IO buffer) - buffer size in kB or MB (for example: `rows=k512` (512 kB buffer) or `rows=m20` (20MB buffer))
`ns=<nullstring>`	How odb represents NULL values in the output file. Default is the empty string (two field separators one after the other)
`es=<emptystring>`	How odb represents VARCHAR empty strings (NOT NULL with zero length) values in the output file. Default is the empty string (two field separators one after the other)
`nomark`	Don’t print the number of records extracted so far by each thread.
`soe`	Stop On Error. odb stops as soon as it encounters an error.
`parallel=num`	odb uses as many *threads triplets* (extract from source, extract from target, compare) as the parallel argument. Each thread will take care of a specific range of the source table data defined through the **splitby** option.
`uncommitted`	odb adds `FOR READ UNCOMMITTED ACCESS` to the select(s) command(s).

Diff Operator	Meaning
splitby=<column>	Lets you to use parallel extract from any database. **<column> has to be a SINGLE, numeric column (or expression)**. odb calculates min()/max() value for <column> and assigns it to each <parallel> thread the extraction of the rows in its *bucket*. **Example** …:splitby=emp_id:parallel=4… with min(emp_id)=1 and max(emp_id)=1000, the four threads will extract the following rows: thread #0 emp_id >=1 and emp_id < 251 thread #1 emp_id >=251 and emp_id < 501 thread #2 emp_id >=501 and emp_id < 751 thread #3 emp_id >=751 and emp_id < 1001 (odb uses max(emp_id) + 1) If the values are not equally distributed, then data extraction is deskewed.
print=[I][C][D]	Specifies which rows are printed in the output file: I prints the new rows on target. (Based on **key**.) D prints the missing rows on target. (Based on **key**.) C prints the source rows with the same **key** columns but differences in other fields. The default value for print is IDC.
pre={@sqlfile}|{[sqlcmd]}	odb runs a **single instance** of either a sqlfile script or sqlcmd SQL command (enclosed between square brackets) on the **target system** immediately before reading the target table.
post={@sqlfile}|{[sqlcmd]}	odb runs a **single instance** of either a sqlfile script or sqlcmd SQL command (enclosed between square brackets) on the **target system** immediately after the target table has been compared.
tpar=num	The number of tables to compare in parallel when you have a list of tables in input.
loaders=num	odb uses num load threads for each extract thread. Default is 2 loaders per extractor.
pwhere=<where condition>	This option is used in conjunction with parallel to *diff* only records satisfying the where condition. For example: you want to compare rows with TRANS_TS > 1999-12-12 09:00:00 from the source table TRAFODION.MAURO.MFORDERS using eight parallel streams to a target table having the same name as the source table: src=trafodion.mauro.mforders:tgt=trafodion.dest_schema.%t:parallel=8:pwhere=[TRANS_TS > TIMESTAMP '1999-12- 12 09:00:00']… You can enclose the where condition between square brackets to avoid a misinterpretation of the characters in the where condition.
quick	Limits the comparison to the columns in the key option (PK by default). This is a fast way to check for new/missing records but it will not find rows with differences in *non-key* columns.
time	odb prints a *timeline*. (Milliseconds from starts)

Chapter 7. odb as a Query Driver (Technology Preview)

7.1. Getting CSV Output

It's often handy to get a CSV output ready to be imported into your spreadsheet while running performance tests. You can easily get this kind of output with `–c` odb option.

Example

```
$ ./odb64luo -u mauro -p xxx -d pglocal -x 3:"select count(*) from tpch.region" \
-f 5:Q01.sql -f 3:Q02.sql -T 4 -q –c
```

This command runs:

- Three copies of the `select count(): -x 3:”select count() from tpch.region”`
- Five copies of `Q01.sql: -f 5:Q01.sql`
- Three copies of `Q02: -f 3:Q02.sql`
- Queuing the resulting 11 executions into four threads: `-T 4`
- Omitting query text and query results (`-q` is equivalent to `–q all`): `-q`
- Printing a CSV output: `–c`

The command produces the following output:

```
odb [2011-12-12 08:08:43]: starting (4) threads...
Thread id,Proc id,Thread Exec#,Script
Cmd#,File,Label,Command,Rows,Rsds,Prepare(s),Exec(s),1st
Fetch(s),Fetch(s),Total(s),STimeline,ETimeline
1,1,0,0,(none),,"select count() from
tpch.region",1,20,0.000,0.109,0.000,0.000,0.109,94,203
0,0,0,0,(none),,"select count() from
tpch.region",1,20,0.000,0.125,0.000,0.000,0.125,94,219
2,2,0,0,(none),,"select count() from
tpch.region",1,20,0.000,0.109,0.000,0.000,0.109,110,219
2,6,1,0,Q01.sql,,"SELECT L_RETURNFLAG,
L_LINESTATUS,SUM(L_QUANTITY)>",4,234,0.000,136.297,0.000,0.000,136.297,141,136438
2,10,2,0,Q02.sql,,"SELECT S_ACCTBAL, S_NAME,
N_NAME,P_PARTKEY,P_MF>",0,274,0.000,0.468,0.000,0.016,0.484,136438,136922
0,4,1,0,Q01.sql,,"SELECT L_RETURNFLAG,
L_LINESTATUS,SUM(L_QUANTITY)>",4,234,0.000,139.667,0.016,0.016,139.683,0,139683
0,8,2,0,Q02.sql,,"SELECT S_ACCTBAL, S_NAME, N_NAME,
P_PARTKEY,P_MFG>",0,274,0.000,0.015,0.000,0.000,0.015,139683,139698
1,5,1,0,Q01.sql,,"SELECT L_RETURNFLAG,
L_LINESTATUS,SUM(L_QUANTITY)>",4,234,0.000,144.347,0.015,0.015,144.362,141,144503
1,9,2,0,Q02.sql,,"SELECT S_ACCTBAL, S_NAME, N_NAME,
P_PARTKEY,P_MFG>",0,274,0.000,0.000,0.000,0.016,0.016,144503,144519
3,3,0,0,Q01.sql,,"SELECT L_RETURNFLAG,
L_LINESTATUS,SUM(L_QUANTITY)>",4,234,0.000,144.394,0.016,0.016,144.410,390,144800
3,7,1,0,Q01.sql,,"SELECT L_RETURNFLAG,
L_LINESTATUS,SUM(L_QUANTITY)>",4,234,0.000,69.373,0.000,0.000,69.373,144800,214173
odb statistics:
        Init timestamp: 2011-12-12 08:08:42
        Start timestamp: 2011-12-12 08:08:43
        End timestamp: 2011-12-12 08:12:17
    Elapsed [Start->End] (s): 214.173
----
```

The CSV output columns have the following meaning:

Column	Meaning
`Thread ID`	Thread ID. Number of threads limited to 4 —> thread id values are 0, 1, 2, 3
`Proc ID`	Execution number. 11 executions in the 0-10 range.
`Thread Exec#`	Progressive number (starting from 0) of execution for a specific thread.
`Script Cmd#`	If your script contains multiple SQL statement, then they are numbered starting from zero.
`File`	Script file name or `(null)` for `-x` commands.
`Label`	The label assigned though `set qlabel` in the scripts.
`Command`	First 30 characters of the SQL command. It will end with > if the command text was truncated.
`Rows`	The number of returned rows. Not printed if you used `-q`.
`Rsds`	Record Set Display Size. Gives you an idea of *how big* the result set is.
`Prepare(s)`	Prepare (compile) time in seconds.
`Exec(s)`	Execution time in seconds.
`1st Fetch(s)`	Time needed to fetch the first row in seconds.
`Fetch(s)`	Total Fetch time in seconds.
`Total(s)`	Total query elapsed time from prepare to fetch in seconds.
`Stimeline`	Queries start time line in milliseconds.
`Etimeline`	Queries end time line in milliseconds.

7.2. Assign Label to a Query

Sometimes it's not easy to recognize a query by reading the first 30 characters. Therefore, odb lets you assign a label to a generic query using:

```
SET QLABEL <label>
```

Example

```
~/Devel/odb $ cat script.sql

-- {project-name} TPC-H Query 1 SET QLABEL Q01
SELECT
   L_RETURNFLAG
 , L_LINESTATUS
 , SUM(L_QUANTITY) AS SUM_QTY
 ...

-- TPC-H/TPC-R Minimum Cost Supplier Query (Q2)
SET QLABEL Q02
SELECT
   S_ACCTBAL
   , S_NAME
   ...
```

Running this script includes the Query Label in the CSV output:

```
~/Devel/odb $ ./odb64luo -u mauro -p xxx -d pglocal -f script.sql -q -c

odb [2011-12-12 09:06:28]: starting (1) threads...
Thread id,Proc id,Thread Exec#,Script
Cmd#,File,Label,Command,Rows,Rsds,Prepare(s),Exec(s),1st
Fetch(s),Fetch(s),Total(s),STimeline,ETimeline
0,0,0,0,script.sql,Q01,"SELECT L_RETURNFLAG, L_LINESTATUS,
SUM(L_QUANTITY)>",4,234,0.000,43.102,0.000,0.000,43.102,0,43102
0,0,0,1,script.sql,Q02,"SELECT S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY,
P_MFG>",0,274,0.000,0.016,0.000,0.000,0.016,43102,43118
odb statistics:
        Init timestamp: 2011-12-12 09:06:28
        Start timestamp: 2011-12-12 09:06:28
        End timestamp: 2011-12-12 09:07:11
    Elapsed [Start->End] (s): 43.118
```

7.3. Run All Scripts With a Given Path

Using `-S <path>` or `-P <path>` options you can run all scripts with a given path (for example, all files in a directory) either serially (`-S`) or in parallel (`-P`).

Both options let you to use ***multiplying factors*** to run all scripts multiple times. This multiplying factors are defined with a `<number>:` preceding the script path.

Examples

odb Command Line	Action
`odb64luo -S ./test/queries/*.sql -c -q`	Executes **serially** all scripts with extension `.sql` under `./test/queries/` providing CSV type output (`-c`) and omitting query output (`-q`).
`odb64luo -P test/queries/* -T 50 -c -q`	Runs **in parallel** all files under `test/queries/` using 50 threads (ODBC connections) (`-T 50`), with CSV output (`-c`) and omitting query output (`-q`).
`odb64luo -P 3: test/queries/* -T 3 -c -q`	Runs **in parallel three times (`3:`)** all files under `test/queries/` using three threads (ODBC connections) (`-T 3`), with CSV output (`-c`) and omitting query output (`-q`). Scripts will be assigned to threads using ***standard assignment***.
`odb64luo -P -3: test/queries/* -T 3 -c -q`	Runs **in parallel three times (`-3:`)** all files under `test/queries/` using three threads (ODBC connections) (`-T 3`), with CSV type output (`-c`) and omitting query output (`-q`). Scripts will be assigned to threads using ***round-robin assignment***.

To understand the difference between **standard** and **round-robin** assignments, imagine you have four scripts in the target path. This is how the executions will be assigned to threads:

	Standard Assignment (es. -P 3:)			Round-Robin Assignment (es. -P -3:)		
	Thread 1	**Thread 2**	**Thread 3**	**Thread 1**	**Thread 2**	**Thread 3**
nth execution	…	…		…	…	
4th execution	`Script4.sql`	`Script4.sql`	…	`Script2.sql`	`Script3.sql`	…
3rd execution	`Script3.sql`	`Script3.sql`	`Script3.sql`	`Script3.sql`	`Script4.sql`	`Script1.sql`
2nd execution	`Script2.sql`	`Script2.sql`	`Script2.sql`	`Script4.sql`	`Script1.sql`	`Script2.sql`
1st execution	`Script1.sql`	`Script1.sql`	`Script1.sql`	`Script1.sql`	`Script2.sql`	`Script3.sql`

7.4. Randomizing Execution Order

You can use the `-Z` option to *shuffle* the odb internal execution table. This way the execution order is not predictable.

Examples

odb Command Line	Action
`odb64luo… -S 3: test/queries/* -Z -c –q`	Executes three times (`3:`) all files in the `test/queries` directory serially (`-S`) and in random order (`-Z`).
`odb64luo… -P 3: test/queries/* -Z –T 5 -c -q`	Executes three times (`3:`) all files in the `test/queries` directory in parallel (`-P`), using five threads (`-T 5`) and in random order (`-Z`).

7.5. Defining a Timeout

You can stop odb after a given timeout (assuming the execution is not already completed) using `-maxtime <seconds>` option.

Example

```
~/Devel/odb $ ./odb64luo -S /home/mauro/scripts/*.sql –maxtime 7200
```

The command executes, **serially,**(all scripts with extension `.sql` under `/home/mauro/scripts/`; if the execution is not completed after two hours (7200 seconds), then odb stops.

7.6. Simulating User Thinking Time

You can simulate user ***thinking time*** using the `-ttime <delay>` option. This argument introduces a `<delay>` millisecond pause between two consecutive executions in the same thread.

Example

```
~/src/C/odb $ ./odb64luo -f 5:script1.sql -c -q -ttime 75 -T 2
```

This command runs five times `script1.sql` using two threads. Each thread waits 75 milliseconds before starting the next execution within a thread. You can also use a ***random thinking time*** in a given `min:max` range.

Example

The following command starts commands within a thread with a random delay between 50 and 500 milliseconds:

```
~/src/C/odb $ ./odb64luo -f 5:script1.sql -c -q -ttime 50:500 -T 2
```

7.7. Starting Threads Gracefully

You might want to wait a little before starting the next thread. This can be obtained using the `-delay` option.

Example

```
~/src/C/odb $ ./odb64luo -f 5:script1.sql -c -q -delay 200 -T 2
```

This command runs five times `script1.sql` using two threads. Each thread will be started 200 milliseconds after the other.

`-delay` introduces a delay during threads start-up while `–ttime` introduces a delay between one command and another within the same thread.

7.8. Re-looping a Given Workload

Using `-L` option you can re-loop the workload defined through `-x`, `-f`, `-P`, and `-S` commands a given number of times. Each thread will re-loop the same number of times.

Example

```
~/src/C/odb $ *./*odb64luo -f 5:script1.sql -c -q -M 75 -T 2 -L 3
```

re-loops three times (`-L 3`) the same five executions, using two threads (`-T 2`) with a 75 millisecond pause (`-M 75`) between two consecutive executions in the same thread.

Chapter 8. odb as a SQL Interpreter (Technology Preview)

To start the odb SQL Interpreter you have to use `-I` (uppercase i) switch with an optional argument.

Example

```
$ odb64luo –u user –p xx –d dsn –I MFTEST
```

The optional `-I` argument (`MFTEST` in this example) is used to specify the **`.odbrc`** section containing commands to be automatically executed when odb starts. See <sql_run_commands, Run Commands When Interpreter Starts>>.

8.1. Main odb SQL Interpreter Features

1. **It uses `mreadline` library** to manage command line editing and history. History will keep track of the whole **command**, not just… lines: if you enter a SQL command in more than one line:

   ```
   S01_Maurizio@TRAFODION64[MFTEST]SQL> select
   S01_Maurizio@TRAFODION64[MFTEST]...> count()
   S01_Maurizio@TRAFODION64[MFTEST]...> from
   S01_Maurizio@TRAFODION64[MFTEST]...> t1;
   ```

   When you press the up arrow key the whole command (up to semi-colon) will be ready for editing and/or re-run.

   `mreadline` provides several useful extra features:

   - **CTRL-V** to edit the current command using your preferred editor (`$EDITOR` is used). When the editing session is closed the current command is automatically updated.
   - **CTRL-U/CTRL-L** to change the command case.
   - **CTRL-X** to kill the current command.
   - See on-line help for the other `mreadline` commands.

2. **History is saved** when you exit the SQL Interpreter in a file identified by the `ODB_HIST` environment variable. You can change the number of commands saved in the history file (default `100`):

```
S01_Maurizio@TRAFODION64[MFTEST]SQL> SET HIST 200
```

3. **Customizable prompt**. You can personalize your prompt through the set prompt command. Under Unix/Linux/Cygwin you can use the standard ANSI codes to create color prompts. See Customize Interpreter Prompt.

4. **Multi-threaded odb instances** can be run from within the single-threaded Interpreter with the `odb` keyword. This runs another odb instance using the same credentials, data source, and connection attributes used to start the interpreter:

```
S01_Maurizio@TRAFODION64[MFTEST]SQL> odb -l
src=myfile:tgt=mytable:parallel=8:...

S01_Maurizio@TRAFODION64[MFTEST]SQL> odb -e
src=mytable:tgt=myfile:parallel=8:...
```

5. **Define Aliases** with parameter substitution.

   Example

```
root@MFDB[MFDB]SQL> SET ALIAS count "SELECT ROW COUNT FROM &1;"
```

   When you call the alias `count` the first argument will be substituted to `&1`. You can use **up to nine** positional parameters (`&1` to `&9`).

6. You can **run operating system commands** with `!command`.

7. You can run scripts with `@script`.

8. You can spool to file with `set spool <myfile>` and stop spooling with `set spool off`.

9. You can switch to a special *prepare only* mode with `set prepare on`. This way, commands you type will be just prepared, not executed.

10. Different databases use different commands to set default schema(s):

    - Trafodion: `set schema <name>;`
    - MySQL: `use <name>;`
    - PostgreSQL/Vertica: `set search_path to <name1,name2,…>;`
    - Teradata: `set database <name>;`

    `set chsch <command>` is used to define database specific commands to change your schema. When odb recognize the `change schema` command it will update accordingly internal catalog (if any) and schema names.

11. To list database objects, you can use `ls` command.

    Examples

```
S01_Maurizio@MFTEST[MFTEST]SQL> ls . # list all objects in the current

schema
TABLE : CITIES
TABLE : CUSTOMER
TABLE : LINEITEM
TABLE : NATION
TABLE : ORDERS
TABLE : PART
TABLE : PARTSUPP
TABLE : REGION
TABLE : SUPPLIER
TABLE : T1
VIEW : V_CITIES

S01_Maurizio@MFTEST[MFTEST]SQL> ls -t %S << list tables (-t) ending with S CITIES
ORDERS
S01_Maurizio@MFTEST[MFTEST]SQL> ls -v << list views (-v) V_CITIES
S01_Maurizio@MFTEST[MFTEST]SQL> ls -s << list schemas (-s)

... and so on ...
```

12. To get tables DDL, you can use either `ls –T <table>` or `ls -D <table>`.

    Examples

```
mauro pglocal[PUBLIC] (09:12:56) SQL> ls -T tpch.orders

Describing: postgres.TPCH.orders
+---------------+-------------+----+-------+---------------+
|COLUMN         |TYPE         |NULL|DEFAULT|INDEX          |
+---------------+-------------+----+-------+---------------+
|o_orderkey     |int8         |NO  |       |orders_pkey 1 U|
|o_custkey      |int8         |NO  |       |               |
|o_orderstatus  |bpchar(1)    |NO  |       |               |
|o_totalprice   |numeric(15,2)|NO  |       |               |
|o_orderdate    |date         |NO  |       |               |
|o_orderpriority|bpchar(15)   |NO  |       |               |
|o_clerk        |bpchar(15)   |NO  |       |               |
|o_shippriority |int4         |NO  |       |               |
|o_comment      |varchar(80)  |NO  |       |               |
+---------------+-------------+----+-------+---------------+

mauro pglocal[PUBLIC] (09:13:20) SQL> ls -D tpch.orders
CREATE TABLE postgres.TPCH.orders ( o_orderkey int8
,o_custkey int8
,o_orderstatus bpchar(1)
,o_totalprice numeric(15,2)
,o_orderdate date
,o_orderpriority bpchar(15)
,o_clerk bpchar(15)
,o_shippriority int4
,o_comment varchar(80)
,primary key (o_orderkey)
);
```

13. You can **define your own variables** or use odb internal variables or environment variables directly from the Interpreter.

14. You can `set pad fit` to ***automatically shrink CHAR/VARCHAR fields in order to fit one record in one line***. Line length is defined through `set scols #`. Each record will be printed in one line truncating the length of CHAR/VARCHAR fields proportionally to their original display size length. In case of field truncation a > character will be printed at the end of the truncated string.

    Example

```
MFELICI [MAURIZIO] (03:30:32) SQL> select [first 5]  from part;

P_PARTKEY|P_NAME                          |P_MFGR         |P_BRAND|P_TYPE         |
P_SIZE   |P_CONTAINER|P_RETAILPRICE|P_COMMENT
---------+------------------------------+---------------+-------+---------------+
----------+-------+---------------+-------------
33       |maroon beige mint cyan peru    |Manufacturer#2>|Brand#>|ECONOMY PLATED>|
16|LG PKG>|          933.03|ly eve
39       |rose dodger lace peru floral   |Manufacturer#5>|Brand#>|SMALLPOLISHED> |
43|JUMBO >|          939.03|se slowly abo>
60       |sky burnished salmon navajo hot|Manufacturer#1>|Brand#>|LARGE POLISHED>|
27|JUMBO >|          960.06| integ
81       |misty salmon cornflower dark f>|Manufacturer#5>|Brand#>|ECONOMY BRUSHE>|
21|MED BA>|          981.08|ove the furious
136      |cornsilk blush powder tan rose |Manufacturer#2>|Brand#>|SMALL PLATED S>|
2|WRAP B>|         1036.13|kages print c>
```

15. You can `set plm` to print one field per row. This is useful when you have to carefully analyze few records.

    Example

```
MFELICI [MAURIZIO] (03:38:12) SQL> SET PLM ON
MFELICI [MAURIZIO] (03:38:12) SQL> select * from part where p_partkey =136;

P_PARTKEY    136
P_NAME      :cornsilk blush powder tan rose P_MFGR :Manufacturer#2
P_BRAND     :Brand#22
P_TYPE      :SMALL PLATED STEEL
P_SIZE       2
P_CONTAINER :WRAP BAG P_RETAILPRICE:1036.13
P_COMMENT   :kages print carefully
```

16. Check the rest on your own.

8.1.1. odb SQL Interpreter help

```
`mauro pglocal[PUBLIC] (06:51:20) SQL>` *help

All the following are case insensitive:
  h | help                : print this help
  i | info                : print database info
  q | quit                : exit SQL Interpreter
  c | connect { no | [user[/pswd][;opts;&#8230;] (re/dis)connect using previous or new
user
  odb odb_command         : will run an odb instance using the same DSN/credentials
  ls -[type] [pattern]    : list objects. Type=(t)ables, (v)iews, s(y)nonyns, (s)chemas
                          : (c)atalogs, syst(e)m tables, (l)ocal temp, (g)lobal temp
                          : (m)at views, (M)mat view groups, (a)lias, (A)ll object
types
                          : (D)table DDL, (T)table desc
  print <string>          : print <string>
  !cmd                    : execute the operating system cmd
  @file [&0]...[&9]       : execute the sql script in file
  set                     : show all settings
  set alias [name] [cmd|-]: show/set/change/delete aliases
  set chsch [cmd]         : show/set change schema command
  set cols [#cols]        : show/set ls number of columns
  set cwd [<directory>]   : show/set current working directory
  set drs [on|off]        : show/enable/disable describe result set mode
  set fs [<char>]         : show/set file field separator
  set hist [#lines]       : show/set lines saved in the history file
  set maxfetch [#rows]    : show/set max lines to be fetched (-1 = unlimited)
  set nocatalog [on|off]  : show/enable/disable "no catalog" database mode)
  set nocatnull [on|off]  : show/enable/disable "no catalog as null" database mode)
  set noschema [on|off]   : show/enable/disable "no schema" database mode)
  set nullstr [<string>]  : show/set string used to display NULLs ( to make it Null)
  set pad [fit|full|off]  : show/set column padding
  set param name [value|-]: show/set/change/delete a parameter
  set pcn [on|off]        : show/enable/disable printing column names
  set plm [on|off]        : show/enable/disable print list mode (one col/row)
  set prepare [on|off]    : show/enable/disable 'prepare only' mode
  set prompt [string]     : show/set prompt string
  set query_timeout [s]   : show/set query timeout in seconds (def = 0 no timeout)
  set quiet [cmd|res|all|off] : show/enable/disable quiet mode
  set rowset [#]          : show/set rowset used to fetch rows
  set soe [on|off]        : show/enable/disable Stop On Error mode
  set spool [<file>|off]  : show/enable/disable spooling output on <file>
  <SQL statement>;        : everything ending with ';' is sent to the database
mreadline keys:
  Control-A  : move to beginning of line      Control-P  : history Previous
  Control-E  : move to end of line            Up Arrow   : history Previous
  Control-B  : move cursor Back               Control-N  : history Next
  Left Arrow : move cursor Back               Down Arrow : history Next
  Control-F  : move cursor Forward            Control-W  : history List
  Right Arrow: move cursor Forward            Control-R  : Redraw
  Control-D  : input end (exit) - DEL right   Control-V  : Edit current line
  Control-L  : Lowercase Line                 Control-X  : Kill line
  Control-U  : Uppercase Line #               Control-G  : load history entry #
```

8.2. Run Commands When the Interpreter Starts

When the odb SQL Interpreter starts it looks for the ***Initialization File***. This Initialization File is made of ***Sections*** containing the commands to be executed.

To find the Initialization File, odb checks the `ODB_INI` environment variable. If this variable is not set, then odb looks for a file named `.odbrc` (*nix) or `_odbrc` (Windows) under your HOME directory.

The ***Initialization File*** contains ***Sections*** identified by names between square brackets. For example, the following section is named `MFTEST`:

```
[MFTEST]
set pcn on
set pad fit
set fs |
set cols 3 30
set editor "vim -n --noplugin"
set efile /home/felici/.odbedit.sql set prompt "%U %D [%S] (%T) %M> "
set alias count "select row count from &1;"
set alias size "select sum(current_eof) from table (disk label statistics (&1) );"
set alias ll "select left(object_name, 40) as object_name, sum(row_count) as nrows,
count(partition_num) as Nparts, sum(current_eof) as eof from table(disk label
statistics(
using (select  from (get tables in schema &catalog.&schema, no header, return full
names)
s(b) ))) group by object_name order by object_name;"
set schema TRAFODION.MAURIZIO;
```

the odb SQL Interpreter automatically runs all commands in the section identified by the `-I` argument (for example `-I MFTEST`). A section named `DEFAULT` will be executed when `-I` has no arguments.

8.3. Customizing the Interpreter Prompt

You can define your prompt through the `set prompt` command when running the SQL Interpreter. `set prompt` can be executed interactively or included in your (`$ODB_INI`) ***Initialization File***. `set prompt` recognizes and expands the following variables:

- **`%U`** —> User name
- **`%D`** —> Data Source name
- **`%S`** —> Schema name
- **`%T`** —> Current Time
- **`%M`** —> odb mode:

  `SQL` when running sql commands

  `PRE` if you're in "prepare only" mode

  `SPO` if you are spooling output somewhere

  `NDC` (No Database Connection)

Example

```
SET PROMPT "Prompt for %U connected via %D to %S in %M mode > "
```

Generates the following prompt:

```
Prompt for S01_Maurizio connected via CIV to CIV03 in SQL mode >
```

Under Cygwin, Unix and Linux (and probably under Windows too using ANSI.SYS driver - not tested), you can use standard ANSI escape color codes.

Example

```
set prompt "\^A^[[01;32m\^A%U@%D^A\^[[01;34m^A[%S]\^A^[[00m\ ^A (%T) %M> "
```

Where:

1. **^A** is a *real* Control-A (ASCII 001 and 002) before and after each color code sequence.
2. **^[** is a *real* Escape Character. The meaning of the ANSI color codes are:

   ^[[01;32m —> green

   ^[[01;34m —> blue

   ^[[00m —> reset.

Example Prompt

Chapter 9. Appendixes

9.1. A. Troubleshooting

1. odb uses Condition Variables to synchronize threads during copy and parallel load operations.
2. Most of the memory allocation operations are dynamic. For example, you can execute an SQL command as long as you want. However, you can *hard code* limits as follows:

```
#define MAX_VNLEN   32 /* Max variable name length */
#define MAXCOL_LEN 128 /* Max column name length */
#define MAXOBJ_LEN 128 /* Max catalog/schema/table name length */
#define MAX_CLV     64 /* Max command line variables (-var) */
```

3. Some Linux/UNIX systems (notably the Linux Loader) have huge default stack size. Due to this extremely large value, you can have errors like this when starting tens/hundreds of threads:

```
Error starting cmd thread #: cannot allocate memory
```

If you get this error, then check your default stack size:

```
$ ulimit -a
core file size                          (blocks, -c) 0
data seg size                           (kbytes, -d) unlimited
max nice                                        (-e) 0
file size                               (blocks, -f) unlimited
pending signals                                 (-i) 137215
max locked memory                       (kbytes, -l) 32
max memory size                         (kbytes, -m) unlimited
open files                                      (-n) 65536
pipe size                            (512 bytes, -p) 8
POSIX message queues                     (bytes, -q) 819200
max rt priority                                 (-r) 0
stack size                              (kbytes, -s) 204800
cpu time                               (seconds, -t) unlimited
max user processes                              (-u) 2047
virtual memory                          (kbytes, -v) unlimited
file locks                                      (-x) unlimited
```

In the example above, the `stack size` value is the problem. Reset it to a reasonable value. (The value will be reset to the initial value when you start a new session).

Example

```
$ ulimit -s 4096
```

4. If you get errors such as:

```
C:\Users\felici> odb64luo -u xx -p yy -d oraxe -l
src=region.tbl.gz:tgt=region:fs=^|:truncate

odb [2012-10-11 13:27:22]: starting ODBC connection(s)... 0
[0] odb(5020) - [Oracle][ODBC]Optional feature not implemented. (State: HYC00 Native
Err: 0)
```

   Try adding `-nocatnull` to your command line. When the back-end database doesn't use catalogs, then you should use an **empty string** as catalog name.

   

   Some flawed ODBC Drivers unfortunately want NULL here — instead of **empty strings** as it should be.

5. You can have errors loading `TIME(N)` fields with` N>0` under Trafodion because the ODBC Driver does not manage the field display size when `N>0`.
6. If you have problems starting odb on Unix/Linux check:
   - The shared library dependencies with `ldd <odb_executable_name>`.
   - The shared lib path defined in the following environment variables used by the shared library loader:
     - **Linux**: `LD_LIBRARY_PATH`
     - **IBM IAX**: `LIBPATH` (not currently supported)
     - **HP/UX**: `SHLIB_PATH` (not currently supported)

9.2. B. Develop and Test odb

9.2.1. Develop

odb is coded in "ANSI C" (K&R programming style) and is compiled in a 64-bit version on the Linux platform, linked to the unixODBC driver manager. Other platforms, compilers, and ODBC libraries have not yet been tested.

Platform	Compiler	ODBC Libraries	Note
Linux	gcc	unixODBC (supported), iODBC (not currently supported), Data Direct (not currently supported), Teradata (not currently supported)	64 bit (32 bit is not currently supported)
MS-Windows (not tested)	Visual Studio (not tested)	MS-Windows (not tested)	64 bit

C compilers are set with “all warnings” enabled and odb has to compile, on each platform, with no errors (of course) AND no warnings. Tools used to code odb:

- **vim** (http://www.vim.org) as editor (or Visual Studio embedded editor)
- **splint** (http://www.splint.org) to statically check the source code

9.2.2. Test

The info, load, extract, and copy operations of odb have been fully tested. In addition, odb had been tested using a set of 137 standard tests to check functionalities and identify memory/thread issues.