Trafodion odb User Guide

Trafodion Release 1.3—April 2015

Published: November 2015
Edition: Trafodion Release 1.3

Legal Notice

Table of Contents

About This Document

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

> ***Note:*** *In Trafodion Release 1.3, 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.*

Intended Audience

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

New and Changed Information in This Edition

This manual is new.

Document Conventions

The manual uses the following typographic conventions:

Convention	Description	Example
Monospace dark gray	Command prompt	`mfelici ~odb $`
Monospace orange	Typed command	`./odb64luo -h`
Monospace blue	Command output	`[0.0.0]--- 1 row(s) selected in 0.015s`
Monospace gray	Comments and clarifications	`# this will drop all views in your schema`

Publishing History

Product Version	Publication Date
Trafodion Release 1.3.0	November 2015

We Encourage Your Comments

The Trafodion community encourages 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:

issues@trafodion.incubator.apache.org

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

1 Introduction

1.1 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. In Trafodion Release 1.3, 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.

2 Installation and Configuration

2.1 odb Requirements

A general requirement is the ODBC library for the database you want to connect to and:

Platform	Requirements
Linux	**pthread libraries** (Generally installed by default)
Windows (not yet tested)	**Microsoft Visual C++ 2010 Redistributable Package (x86)**

> ***Note:*** *In Trafodion Release 1.3, odb is available only for the Linux platform.*

2.2 Installing and Configuring the Required unixODBC

This section explains how to install and configure **unixODBC**, which is required for Trafodion Release 1.3.

1. Grab the source code tarball from http://www.unixodbc.org. Use at least version 2.3.*x*.
2. Unpack the tarball:
   `$ tar xzvf unixODBC-2.3.1.tar.gz`
3. Configure unixODBC installation:
   `$ cd unixODBC-2.3.1`
   `$ ./configure --disable-gui --enable-threads --disable-drivers`
   This will install unixODBC under /usr/local and requires root access. If you don't have root privileges or you want to install unixODBC somewhere else you have to add `--prefix=<installation_path>` to the configure command here above. For example:
   `$ ./configure --prefix=/home/mauro/uodbc --disable-gui --enable-threads --disable-drivers`
4. Compile unixODBC sources:
   `$ make`
5. Install unixODBC:
   `$ make install`

Now, in order to configure unixODBC, we have to:

1. Define a few environment variables.
2. Define our Data Sources.

Let's start with the environment variables (which you can add to your profile script):

- First you have to set the ODBCHOME variable to the unixODBC installation dir (the one configured via `--prefix` here above). For example:
  `export ODBCHOME=/home/mauro/uodbc`
- Then you have to configure the system data sources directory (the one containing odbc.ini and odbcinst.ini). Normally this is the etc/ directory under $ODBCHOME:
  `export ODBCSYSINI=${ODBCHOME}/etc`
- Then you have to configure the ODBCINI variable to the full path of the odbc.ini file:
  `export ODBCINI=${ODBCSYSINI}/odbc.ini`
- Finally… do not forget to add unixODBC lib directory to your LD_LIBRARY_PATH (Linux) or LIBPATH (IBM AIX) or SHLIB_PATH (HP/UX):
  `export LD_LIBRARY_PATH=${ODBCHOME}/lib`

Now we have to configure our data sources in **odbc.ini**:

```
[<DATA_SOURCE_NAME>]
Description = DSN Description
Driver = <odbcinst.ini corresponding section>
...
Other (Driver specific) parameters
...
```

and **odbcinst.ini**:

```
[<Driver name in odbc.ini>]
Description = Driver description
Driver = <ODBC driver
FileUsage = 1
UsageCount = 1
```

> ***Note**: The Trafodion ODBC driver requires an environment variable, AppUnicodeType, to be specified in odbcinst.ini. This variable must be set to utf16.*
>
> *If you are using Vertica, the Vertica ODBC driver requires an additional section named [Driver] in odbc.ini with specific settings for unixODBC (odbcinst library location and UTF level supported by unixODBC). See the Vertica Programmer's Guide – Additional ODBC Driver Configuration Settings. See also the following example:*

Example:

```
$ cat odbc.ini
[ODBC Data Sources]
VMFELICI                    = VerticaMachine1
traf                        = Trafodion database

[VMFELICI]
Description                 = Vertica Machine 1
Driver                      = VODBC
Database                    = vertica01_machine
Servername                  = server_name
UID                         =
PWD                         =
Port                        = 1111
ConnSettings                =
DriverStringConversions    = NONE
SSLKeyFile                  = /<dir-name>client.key
SSLCertFile                 = /<dir-name>/client.crt

[Driver] << This section is required by Vertica's ODBC Driver
Locale                      = en_US
ODBCInstLib                 = /<dir-name>/uodbc/lib/libodbcinst.so
ErrorMessagesPath           = /opt/vertica/lib64/
DriverManagerEncoding       = UTF-16
LogPath                     = /tmp
LogNameSpace                =
LogLevel                    = 0

[traf]
Description                 = traf DSN
Driver                      = Trafodion
Catalog                     = TRAFODION
Schema                      = QA
DataLang                    = 0
FetchBufferSize             = SYSTEM_DEFAULT
Server                      = TCP:<server-name>:<port-no>
SQL_ATTR_CONNECTION_TIMEOUT = SYSTEM_DEFAULT
SQL_LOGIN_TIMEOUT           = SYSTEM_DEFAULT
SQL_QUERY_TIMEOUT           = NO_TIMEOUT
ServiceName                 = TRAFODION_DEFAULT_SERVICE


$ cat odbcinst.ini
[VODBC]
Description          = Vertica ODBC Driver
Driver               = /<dir-name>/vodbc/lib64/libverticaodbc.so
FileUsage            = 1
UsageCount           = 1

[Trafodion]
Description         = Trafodion ODBC Stand Alone Driver
Driver              = /<dir-name>/conn/clients/odbc/libtrafodbc_drvr64.so
```

```
FileUsage           = 1
```

```
UsageCount          = 1
AppUnicodeType      = utf16

[ODBC]
Threading            = 1
Trace                = Off
Tracefile            = uodbc.trc
```

Another important entry in odbcinst.ini is `Threading`. The following comment (extracted from unixODBC sources `DriverManager/ handles.c`) explain its meaning:

```
 * ...
 * If compiled with thread support the DM allows four different
 * thread strategies
 *
 * Level 0 - Only the DM internal structures are protected
 * the driver is assumed to take care of it's self
 *
 * Level 1 - The driver is protected down to the statement level
 * each statement will be protected, and the same for the connect
 * level for connect functions, note that descriptors are considered
 * equal to statements when it comes to thread protection.
 *
 * Level 2 - The driver is protected at the connection level. only
 * one thread can be in a particular driver at one time
 *
 * Level 3 - The driver is protected at the env level, only one thing
 * at a time.
 *
 * By default the driver open connections with a lock level of 0,
 * drivers should be expecetd to be thread safe now.
 * this can be changed by adding the line
 *
 * Threading = N
 *
 * to the driver entry in odbcinst.ini, where N is the locking level
 * (0-3)
```

2.3 Installing odb

See the *Trafodion Client Installation Guide* (1.3) for details.

3 Basic Concepts

3.1 Getting Help

The following command shows the odb help:

```
mfelici ~/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 Trafodion 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 compressione 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 Trafodion 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
```

3.2 Connecting to a 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 `-ca` command line option.

> ***Note**: 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	

> ***Note**: Command-line options take precedence over environment variables.*

3.3 Listing Available ODBC Drivers and Data Sources

You can list available drivers with `-lsdrv`:

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

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

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

3.4 Obtaining Database Information

Using the `-i` option you can 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:

```
ncsi@ldratc16 ~/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)
```

Listing Database Objects

In the previous section, we have 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

Type	Meaning
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

Examples:

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:

```
ncsi@ldratc16 ~/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

3.5 Performing 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.

Examples:

Example	Action

`delete from &t:MF%`	Purge ALL tables (t:) staring with "MF"
`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.

3.6 Running Commands and Scripts

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

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

For example:

```
felici ~/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.

```
felici ~/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;
felici ~/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,37734107.00,56586554400.73,53758257134.8700,55909065222.827692,25.522006,38273.129735,0.049985,14
78493
...
R,F,37719753.00,56568041380.90,53741292684.6040,55889619119.831932,25.505794,38250.854626,0.050009,14
78870
[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. So, for example, `-q cmd` won't print the ***commands*** being executed:

```
felici ~/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,37734107.00,56586554400.73,53758257134.8700,55909065222.827692,25.522006,38273.129735,0.049985,14
78493
...
R,F,37719753.00,56568041380.90,53741292684.6040,55889619119.831932,25.505794,38250.854626,0.050009,14
78870
[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` won't print the ***results***:

```
felici ~/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)
```

And finally `-q all` (or just `-q`) won't print neither the ***commands*** nor the ***results***:

```
felici ~/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.

> ***Note:*** *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***. For 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 -
```

3.7 Using "here document" Syntax in Shell Scripts

We often need to *embed* SQL commands in shell scripts. In these cases you can use "`-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
```

3.8 Running Multiple Commands and Scripts in Parallel

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. So, for example:

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

will use 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. So, for example, the following command will run 3 instances of " `select count(*) from types` ", 5 instances of "`script1.sql`" and 3 instances of "`script2.sql`" in parallel using 3 + 5 + 3 = 11 threads in total:

```
felici ~/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 number highlighted in red here above is the ***thread ID***. Thread IDs are assigned by odb starting from zero.

You can limit the maximum number of threads with `-T` option. The following example will run the same 11 commands/scripts limiting the number of threads (**and ODBC connections**) to 4:

```
felici ~/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)
```

As you can see the thread ID is now in the 0-3 range because the 11 ***executions*** were ***queued*** into 4 threads. As we will see in the next sessions odb offers several alternatives to queue M executions in N (<M) threads.

3.9 Limiting the Number of Threads Created by odb

By default odb will create as many threads as the numbers of executions. So, the following command:

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

It will create 1 + 3 + 3 = 7 threads. Each thread will start its own ODBC connection.

You can limit the max number of threads using `-T`. So the following:

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

It will create just two threads to execute the seven commands/scripts.

odb will never create more threads than needed:

```
felici ~/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) then needed (5).
```

3.10 Changing the Number of Executions Distributed Across Threads

By defaults executions are distributed in round-robin across threads. Let's say we run:

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

Then, the execution queue will be the following:

	Thread 1	Thread 2	Thread3
3rd execution	mysqlcmd		
2nd execution	Script2.sql	mysqlcmd	mysqlcmd
1st execution	Script1.sql	Script2.sql	Script2.sql

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

- `-Z` (shuffle). This will option will ***randomize*** the execution order
- ***factor sign*** with `-P` option. See Running All Scripts With a Given Path.
- `-dlb` (Dynamic Load Balancing). See Learning How Dynamic Load Balancing Works.

3.11 Learning How Dynamic Load Balancing Works

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

	Thread 1	Thread 2
2nd execution	Script1.2	Script2.2
1st 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 Loading Binary Files), we might want to keep all threads busy at any given time. In these cases we 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.

3.12 Using 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.

So, for example, if you have a script like this:

```
felici ~/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:

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

You will get the following result:

```
felici ~/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. s
2,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)
```

3.13 Understanding Thread ID, Thread Execution#, and Script Command#

Suppose we have a script containing two commands:

```
mauriziof@traf64-dl2a ~/odb $ cat script.sql
SELECT COUNT(*) FROM ORDERS;
SELECT COUNT(*) FROM SUPPLIER;
```

If we run this script ten times using two threads, we will get:

```
mauriziof@traf64-dl2a ~/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***. As we have two threads in our example this ID is either 0 or 1;
2. The second digit is the ***execution#*** for a given thread. As we have ten script executions for two threads, each thread will have to execute this script five times. So the execution#, in our example, is between 0 and 4;
3. The third (last) digit is the ***command#*** in a given script. Our script contains two commands so this value is 0 or 1.

So, for example, **`[0.3.1]`** means that the **first thread** (thread id=0) was executing its **fourth job** (thread execution#=3) and - more specifically - the **second command** in that script (script command#=1).

3.14 Checking 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.

3.15 Using 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. So, for 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. So, for 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.

3.16 Formatting 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 will get a table format like this:

```
$ ./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
```

3.17 Extracting Tables' DDL

You can extract (multiple) tables' DDL using either -i D... or -i U... options. For 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)
);
```

Sometimes you might want to extract the DDL for multiple objects, In this case you can use % wildcard. For 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)
```

When porting DDLs from one database to another we should consider possible differences in text column length semantic: certain database use “character oriented” text columns length while other 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]. For 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)
```

That command will multiply by 2 the length of “non-wide” text fields and by 4 the length of wide text fields.

4 Loading, Extracting, and Copying Data

4.1 Loading Files

You can load a data file using -l option. For 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 will:

- load 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 rowset buffer (rows=1000)
- using UPSERT USING LOAD syntax to achieve better throughput as described in https://wiki.trafodion.org/wiki/index.php/Data_Loading (Trafodion only)
- truncating the target table before loading (truncate)
- using 4 parallel threads to load the target table (parallel=4)

This is a complete list of the 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 will be expanded to the (lower case) table name %T will be expanded to the (upper case) table name %s/%S will be expanded to the schema name %c/%C will be expanded to the catalog name stdin will 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.

Load option	Meaning
map=<mapfile>	This will use mapfile to map source file to target table columns (see paragraph 4.2)
skip=num	This will skip a given number of lines when loading. This can be useful to skip headers in the source file.
max=num	This is the max number of records to load. Default is to load all records in the input file
ns=<nullstring>	odb will insert NULL when it finds nullstring in the input file. By default the nullstring is the empty string
sq=<char>|<code>	This is the string qualifier character used to enclose strings. You can define the escape character the same way as the field separator.
ec=<char>|<code>	This is 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: - 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)) Default value is 100.
bad=[+]file	Where to write rejected rows. If you omit this parameter rejected rows will be 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 will **append** instead of **create** the "bad file".
truncate	Will truncate the target table before loading
ifempty	Will load the target table only if it contains no records
norb	Will load WITH NO ROLLBACK
nomark	Won't print the number of records loaded so far during loads
soe	Stop On Error. Will stop as soon as odb will encounter an error
parallel=num	Number of loading threads. odb will use: - 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 will print up to num error messages per rowset. Normally used with soe to limit the number of error messages printed to the standard error
commit=auto|end|#rows|x#rs	This defines how odb will commit the inserts. You have three choices: - auto will commit every single insert (see also rows load operator). This is the default. - end will commit when all rows (assigned to a given thread) have been inserted - #rows will commit every #rows inserted rows - x#rs will commit every #rs rowsets (see :rows)
direct	This will add /*+ DIRECT */ hint to the insert statement. To be used with Vertica databases in order to store inserted rows "directly" into the ROS (see Vertica's documentation)

Load option	Meaning
fieldtrunc={0-4}	This defines how odb will manage fields longer than the destination target column: - fieldtrunc=0 (default): will truncate 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: print 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. Be aware: last two options could bring to unwanted resilts because, for example, an input string like "2001-10-2345" will be loaded as a valid 2001-10-23 if the target field is a DATE.
em=<char>|<code>	This is the character used to embed binary files (see paragraph 4.6). You can define the embed character the same way as the field separator. No default value.
pre={@sqlfile}|{[sqlcmd]}	odb will run 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 won't be loaded if SQL execution fails and Stop On Error is set
post={@sqlfile}|{[sqlcmd]}	odb will run 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 will load num tables in parallel when src is a list of files to be loaded
show	odb will print what would be loaded in each column but no data is actually loaded. This I useful if you want to see how the input file "fits" into the target tables and is normally used to analyze the first few rows of CSV files (use :max).This option will force: - parallel to 1 - rows to 1 - ifempty to false - truncate to false
maxlen=#bytes	odb will limit the amount of memory allocated in the ODBC buffers for CHAR/VARCHAR fields to #bytes
time	odb will print a "timeline" (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, odb will load node attributes values.
xmlord	By default odb will *match* target table columns with XML node or attributes using their names. If this option is specified odb will load 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	Using this option odb won't load the XML file content. 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 is INSERT. UPSERT and UPSERT USING LOAD are also available for Trafodion.

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

Will truncate and load (CUSTOMER, ORDERS and LINEITEM) tables. Input files have the same name as the target tables – in lower cases). Loads will be distributed among available threads this way:

Load Order	Thread 0	Thread 1	Thread2	Thread3	Thread4
3rd	Read lineitem.tbl	Load TRAFODION.MAURO.LINEITEM	Load TRAFODION.MAURO.LINEITEM	Load TRAFODION.MAURO.LINEITEM	Load TRAFODION.MAURO.LINEITEM
2nd	Read orders.tbl	Load TRAFODION.MAURO.ORDERS	Load TRAFODION.MAURO.ORDERS	Load TRAFODION.MAURO.ORDERS	Load TRAFODION.MAURO.ORDERS
1st	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

> ***Note:*** *You can load gzipped files without any special option. odb will automatically check input files and decompress them on the fly when needed.*

Also note that, even using one single loading thread (parallel=1), odb is faster than without parallel. This because if we do not specify parallel, odb will use 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 ...
```

With parallel=1 we have one thread to read from file and one to write:

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

Now 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.

4.2 Mapping Source File Fields to Target Table Columns

odb, by default, assumes that input files contain as many fields as the target table columns. Also: file fields and target table columns are in the same order. This means that the first field in the input file will be loaded in the first table column, second input field will go 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, 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/timestamp 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[1].
  - `<field>` is one of the following:
    - The ordinal position (starting from zero) of the input file field. So: 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>` will generate/load a string of <LENGTH> characters randomly selected in the following ranges: a-z, A-Z, 0-9
    - `NRAND:<PREC>:<SCALE>` will generate/load a random NUMERIC field with precision `<PREC>` and scale `<SCALE>`
    - `DSRAND:<file>` will select and load a random line from `<file>`
    - `TXTRAND:<MIN_LENGTH>:<MAX_LENGTH>:<file>`: will select and load a random portion of test from `<file>` with length between `<MIN_LENGTH>` and `<MAX_LENGTH>`
    - `LSTRAND:<VALUE1,VALUE2,…>` will select and load a random value from `<VALUE1,VALUE2,…>`
    - `EMRAND:<MIN_ULENGTH>:<MAX_ULENGTH>:<MIN_DLENGTH>:<MAX_DLENGTH>:<SUFFIX1,SUFFIX2,…>` will generate and load 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>`. **Note:** `<START>` starts from zero.
    - `EMPTYASNULL`: will load empty strings in the input file as NULLs (default is to load empty string as… empty strings)
    - `EMPTYASCONST:<CONSTANT>`: will load empty fields in the input file as `<CONSTANT>`
    - `NULL`: will insert NULL

[1] Be aware: this is case sensitive.

- The final, optional, ***transformation operators*** can be:
  - SUBSTR:<START>:<END>. For example if you have an input field containing " Tib:student" a transformation rule like SUBSTR:3:6 will load "Tib" in the database.
  - TSCONV:<FORMAT>. This operator convert 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>. This operator convert dates from the input file format defined through <FORMAT> to YYYY-MM-DD (see TSCONV operator).
    Example: DCONV:B.d.y will convert "August,24 1991" to 1991-08-24
  - TCONV:<FORMAT>. This operator convert times from the input file format defined through <FORMAT> to HH:MM:SS (see TSCONV operator).
  - REPLACE:<READ>:<WRITTEN>. This operator will load the string <WRITTEN> when the input file fields contains <READ>. If the input file string doesn't match <READ> it will be loaded as is (see paragraph 4.3)
  - TOUPPER. This operator converts the string read from the input file to uppercase before loading. Example: proGRAmMEr → PROGRAMMER
  - TOLOWER. This operator converts the string read from the input file to lowercase before loading. Example: proGRAmMEr → programmer
  - FIRSTUP. This operator 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>. This opertator let you to delete or change any character with another. Examples:
    - WORK:7:translit:Gp:HP will load the seventh input field into the target column named WORK and will replace all "G" with "H" and all "p" with "P"
    - WORK:7:translit:Gp\r:HP\d behaves like the previous example but will also delete 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 ".") will result in loading the following values into the target (NAME) column:

```
Maurizio
```

```
Lucia
Giovanni
Antonella
Martina
Marco
```

- COMP. This operator will transform a packed binary COMP into a target database number.
  For example: hex 80 00 00 7b will be loaded as -123
- COMP3:PRECISION:SCALE. This operator will 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 will be loaded as -1234567890123456.789
- ZONED:PRECISION:SCALE. This operator will transform a packed binary ZONED format into a target database number.
  For example: hex 31 32 33 34 35 36 will be loaded as +.123456

4.3 Using mapfiles to Ignore and/or Transform Fields When Loading

Let me use an example to explain 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
```

AGE NAME BDATE

You want to load the highlighted fields into the appropriate column, ***generate*** a unique key for ID and ignore the non-highlighted fields. In addition: you need to ***convert date format*** and replace all occurrencies of "Lucia" with "Lucy". Well, you will need a mapfile like this:

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

Then we will load this way:
$ odb64luo -u user -p xx -d dsn \
       -l src=frends.dat:tgt=TRAFODION.MFTEST.FRIENDS:map=ml1.map:fs=,
```

4.4 Using 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
 └┤  NAME  └┤  BDATE  └┤  JOB
```

You want to load the highlighted 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:

```
felici ~/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
```

Then we will load this way:

```
$ 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.

4.5 Generating and Loading Data

odb can generate and load data for testing purposes. Let's discuss odb capabilities in this area through an example. Suppose you want to fill with test data a table like this:

```
MFELICI [MAURIZIO] (03:37:06) SQL> ls -D person
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 SIGNED(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:

```
felici ~/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` will load a sequence starting from 100 into PID

- `FNAME:DSRAND:datasets/first_names.txt` will load 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` will load LNAME with a random value from last_names.txt
- `COUNTRY:DSRAND:datasets/countries.txt` will load COUNTRY with a random value from countries.txt
- `CITY:DSRAND:datasets/cities.txt` will load CITY with a random value from cities.txt
- `BDATE:DRAND:1800:2012` will generate and load into BDATE a random date between 1800-01-01 and 2012-12-31
- `SEX:LSTRAND:M,F,U` will load SEX with a random value in the M, F, U range
- `EMAIL:EMRAND:3:12:5:8:com,edu,org,net` will generate and load 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` will generate and load a random NUMERIC(9,2)
- `EMPL:DSRAND:datasets/fortune500.txt` will load EMPL with a random value from fortune500.txt
- `NOTES:TXTRAND:20:80:datasets/lorem_ipsum.txt` will load NOTES with a random section of lorem_ipsum.txt with length between 20 and 80 characters
- And, finally, `LOADTS:CTSTAMP` will load the current timestamp into LOADTS

We will generate and load test data with a command like this:

```
$ ./odb64luo -l
src=nofile:tgt=traf.maurizio.person:max=1000000:map=person.map:rows=5000:parallel=8:loadcmd=UL
```

Please note the "`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 1 Ml 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
```

4.6 Loading Default Values

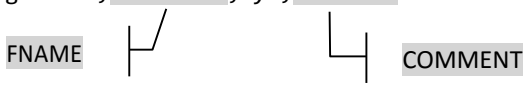
The simpler way to load database generated defaults is to ignore the associated columns in the map file. For example, suppose you have a table like this under Trafodion:

```
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
```



and a map-file like this:

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

Then:

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

4.7 Loading Binary Files

Assuming your backend 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);
```

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

> ***Note:*** *odb will not load rows where the size of the input file is greater than the target database column*

4.8 Reducing the ODBC Buffer Size

odb will allocate memory for the ODBC buffers during load/extract operations based on the max possible length of the source/target columns. So, if you have a column defined as VARCHAR(2000), odb will allocate enough space for 2,000 characters in the ODBC buffer.

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

Suppose we have a table like this:

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

And an input file containing:

```
felici ~/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:

```
felici ~/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:

```
felici ~/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 will allocate the buffer for the max possible length of each field:

```
felici ~/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)
```

4.9 Extracting Tables

You can use odb to extract tables from a database and write them to standard files (or named pipes). This is a table extraction 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 command here above will:

- extract tables REGION, NATION and all tables starting with LIN from TRAFODION.MAURIZIO schema,
- data will be saved into files ext_%t.csv.gz (%t will be expanded to the real table name),
- output file will be gzipped on the fly (uncompressed data will never land to disk),
- text fields will be trimmed
- the IO buffer is 10 MB
- the extraction process will use three threads (ODBC connection)

This is a complete list of the extraction options:

```
-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][: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	This filed is used to define the source table(s). You can use here: - 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>	This is 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 will be expanded to the (lower/upper case) table name %s/%S will be expanded to the (lower/upper case) schema name %c/%C will be expanded to the (lower/upper case) catalog name %d will be expanded to the extraction date (YYYYMMDD format) %D will be expanded to the extraction date (YYYY-MM-DD format) %m will be expanded to the extraction time (hhmmss format) %M will be expanded to the extraction time (hh:mm:ss format) stdout will print the extracted records to the standard output. If you add a +sign in front of the file-name, odb will **append** instead of **create** it. hdfs./<hdfspath>/<file> to write exported table under Hadoop File System
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)
max=num	This is the max number of records to extract. Default is to extract all records

Extract option	Meaning
sq=<char>|<code>	This is the string qualifier character used to enclose strings. You can define the string qualifier the same way as the field separator
ec=<char>|<code>	This is 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 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>	This is how odb will represent NULL values in the output file. Default is the empty string (two field separators one after the other)
es=<emptystring>	This is how odb will represent 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 leading spaces[2] from CHAR fields[3] C trims trailing spaces from CHA R fields3 v trims leading spaces from VARCHAR fields V trims trailing spaces from VARCHAR fields d trims trailing zeros after decimal sign. Example: 12.3000 will be extracted as 12.3. t trims decimal portion from TIME/TIMESTAMP fields. For example: 1999-12-19 12:00:21.345 will be 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 zeros If you do not specify any argument for this operator odb will use "cCvV". In other words :trim: is a shortcut for :trim=cCvV:
nomark	Won't print the number of records extracted so far by each thread
soe	Stop On Error. Will stop as soon as odb will encounter an error
parallel=num	odb will use as many threads as the parallel argument to extract data from partitioned source tables. **You have to use splitby.** Each thread will take 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, odb will start **four** threads (four ODBC connections): - thread 0 will extract partitions 0-7 - thread 1 will extract partitions 8-15

[2] The following characters are considered "spaces": blank, tab, new line, carriage return, form feed, vertical tab

[3] **Note:** When the source table column is defined as NOT NULL and the specific field contains only blanks, odb will leave in the output file one single blank. This will help to distinguish between NULL fields (<field_sep><field_sep>) and NOT NULL fields containing all blanks (<field_sep><blank><field_sep>)

Extract option	Meaning
	- thread 2 will extract partitions 16-23 - thread 3 will extract partitions 24-31
multi	This option can be used in conjunction with parallel operator to write as many ouput 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 will write 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 and will limit 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 8 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 will print up to num error messages per rowset. Normally used with soe to limit the number of error messages printed to the standard error
uncommitted	Will add FOR READ UNCOMMITTED ACCESS to the select(s) command(s)
rtrim	This will 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	It 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%
splitby=<column>	This operator let you to use parallel extract from any database. **<column> has to be a SINGLE, numeric column**. odb will calculate 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 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 data extraction will be deskewed.
pre={@sqlfile|{[sqlcmd]}	odb will run a **single instance** of either 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

Extract option	Meaning

mpre={@sqlfile}|{[sqlcmd]}	Each odb thread will run 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 will run a **single instance** of either sqlfile script or sqlcmd SQL command (enclosed between square brackets) on the **source system** immediately after table extraction.
tpar=num	odb will extract num tables in parallel when src is a list of files to be loaded
maxlen=#bytes	odb will limit the amount of memory allocated in the ODBC buffers for CHAR/VARCHAR fields to #bytes
xml	Will write output file in XML format
time	odb will print a "timeline" (milliseconds from start)

4.10 Extracting a List of Tables

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

```
felici ~/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.

4.11 Copying 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). Using this option data ***will never land to disk***. Target table has to be be created in advance and should have a compatible structure.

Here we have a complete list of the odb's 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]]
```

odb copy's operators:

Copy operator	Meaning
src=<CAT.SCH.TAB>|-file	This filed is to define the source table(s). You can use here: - 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)

Copy operator	Meaning
tgt=<CAT.SCH.TAB>	Target table(s). You can use the following keywords for this field: %t/%T will be expanded to the (lower/upper case) source table name %s/%S will be expanded to the (lower/upper case) source schema name %c/%C will be expanded to the (lower/upper case) source catalog name
sql={[sqlcmd]|@sqlfile|-file}	odb use 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>	This 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	Will truncate the target table before loading
ifempty	Will load the target table only if empty
nomark	Won't print the number of records loaded so far during loads
soe	Stop On Error. Will stop as soon as odb will encounter an error
parallel=num	odb will use as many threads as the parallel argument to extract data from partitioned source tables **PLUS** an equivalent number of threads to write to the target table. For example if you specify parallel=4 and the source table is made of 32 partitions, odb will start **four** threads (four ODBC connections) to read from the source table **PLUS** four threads (four ODBC connections) to write to the target table: - thread 0 will extract partitions 0-7 from source - thread 1 will write data extracted from thread 0 to target - thread 2 will extract partitions 8-15 from source - thread 3 will write data extracted from thread 2 to target - thread 4 will extract partitions 16-23 from source - thread 5 will write data extracted from thread 4 to target - thread 6 will extract partitions 24-31 from source - thread 7 will write data extracted from thread 6 to target **You have to specify splitby.**
pwhere=<where condition>	This option is 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. For example: you want to copy records with TRANS_TS > 1999-12-12 09:00:00 from the source table TRAFODION.MAURO.MFORDERS using 8 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	This defines how odb will commit the inserts. You have three choices: - auto will commit every single insert (see also rows load operator). This is the default. - end will commit when all rows (assigned to a given thread) have been inserted

Copy operator	Meaning
	- #rows will commit every #rows copied rows - x#rs will commit every #rs rowsets copied (see :rows)
direct	This will add /*+ DIRECT */ hint to the insert statement. To be used with Vertica databases in order to store inserted rows "directly" into the ROS (see Vertica's documentation)
errmax=num	odb will print up to num error messages per rowset. Normally used with soe to limit the number of error messages printed to the standard error
uncommitted	Will add FOR READ UNCOMMITTED ACCESS to the select(s) command(s)
splitby=<column>	This operator let you to use parallel copy from any database. **<column> has to be a SINGLE, numeric column**. odb will calculate 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 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 data extraction will be deskewed.
pre={@sqlfile}|{[sqlcmd]}	odb will run 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 won't be loaded if SQL execution fails and Stop On Error is set
mpre={@sqlfile}|{[sqlcmd]}	Each odb thread will run either 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}|{[sqlcmd]}	Each odb thread will run either 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 will run 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 will copy num tables in parallel when src is a list of files to be loaded
loaders=num	odb will use num load threads for each extract thread (default is 2 loaders per extractor)
fwc	Force Wide Characters. odb will consider SQL_CHAR/SQL_VARCHAR fileds as they were defined SQL_WCHAR/SQL_WVARCHAR
bpwc=#	odb internally allocates 4 bytes/char foe 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) will use cdef if SRC/TGT use the same database or char if SRC/TGT databases differ
seq=field#[,start]	odb will add a sequence when loading the target system on column number field#. You can optionally define the sequence start value (default 1)

Copy operator	Meaning
time	odb will print a “timeline” (milliseconds from start)

When copying data from one data source to another, odb will need 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=...
```

4.12 Copying a List of Tables

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

```
felici ~/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`.

Will copy:

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

You can optionally define any other “command line” options in the input file. For example you can use different “splitby columns”:

For example:

```
felici ~/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
```

4.13 Using Case-Sensitive Table and Column Names

The possibility to use case sensitive table/column names depends on your database configuration. odb will maintain table/column case sensitiveness when they are enclosed in double quotes.

This will 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:

```
felici ~/Devel/odb $ ./odb64luo -i T:trafodion.maurizio.\"Names\"
felici ~/Devel/odb $ ./odb64luo -x "select * from trafodion.maurizio.\"Names\""
felici ~/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*.

4.14 Determining the 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. The first option (:tpar) defines how many tables have to be copied/extracted in parallel; the second option defines how many "data streams" to use for each table. This way odb will automatically allocate and start the "right" number of threads.

My **rule of thumb** when copying/loading or extracting tables is to use as many "data streams" as:

min(number of middletier 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

4.15 Integrating With Hadoop

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

1. First Option is to use HIVE (Hadoop DWH) and its ODBC Driver. In this case odb can access HIVE like any other "normal" relational database. So – for example – you can copy to from HIVE and other databases using odb's copy option
2. Second option if to add the **hdfs.** prefix to the input or output file during loads/extracts. In this case the file will be read/written from/to Hadoop. odb interacts directly with the HDFS file system using **libhdfs**. This option is currently available only under Linux.

5 Comparing Tables (Technology Preview)

You can use odb to compare two tables **with the same structure** on different databases. In order to compare the two tables odb will:

1. extract source/target tables ordered by Primary Key or any other set of columns (see key option here below)
2. compare source/target ODBC buffers without "unpacking" them into columns/rows.

Each "comparison stream" is made of three threads:

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

These three threads work in parallel: the "compare" thread will check 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. For example the following command:

```
[dbadmin@n021 mftest]$ odb64luo -u MFELICI:maurizio.felici@hp.com -d MFELICI:VMFELICI -p xx:yy   -diff
src=trafodion.maurizio.lineitem:tgt=mftest.lineitem:key=l_orderkey,l_linenumber:output=lineitem.diff:
rows=m2:print=IDC:splitby=l_orderkey:parallel=8
```

Compare two tables using 8 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).

Here you have an example of the odb output when comparing two tables:

```
[dbadmin@n021 mftest]$ 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
carefully 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,wake carefully 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.

Here we have a complete list of the odb's 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]
```

odb copy's operators:

Diff operator	Meaning
src=<CAT.SCH.TAB>|-file	This field definse the source table(s). You can use here: - 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>	This is the target table.
key=column[,column,...]	This option is used to define how to order records extracted from both source and target table. **If you do not specify any** key **column, odb will use the Primary Key**
output=[+]file	This is the output file where the differences will be reported. You can use stdout to print odb output on the standard output. A **+** sign in front of the file-name will tell odb to **append** to an existing file. Default value: stdout
fs=<char>|<code>	This is the field separator of the output file. 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)

Diff operator	Meaning
rs=<char>|<code>	This is the record separator used in the output file. You can define the record separator the same way as the field separator. Default record separator is '\n' (new line)
max=num	This is the max number of records to compare. Default is to compare all records
rows=<num>|k<num>|m<num>	This 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>	This is how odb will represent NULL values in the output file. Default is the empty string (two field separators one after the other)
es=<emptystring>	This is how odb will represent 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	Won't print the number of records extracted so far by each thread
soe	Stop On Error. Will stop as soon as odb will encounter an error
parallel=num	odb will use 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	Will add FOR READ UNCOMMITTED ACCESS to the select(s) command(s)
splitby=<column>	This operator let you to use parallel extract from any database. **<column> has to be a SINGLE, numeric column (or expression)**. odb will calculate 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 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 data extraction will be deskewed.
print=[I][C][D]	This operator is used to specify which rows will be printed in the output file: 'I' will print the new rows on target (based on **key**) 'D' will print the missing rows on target (based on **key**) 'C' will print the source rows with the same **key** columns but differences in other fields Default value for print is IDC
pre={@sqlfile}|{[sqlcmd]}	odb will run a **single instance** of either sqlfile script or sqlcmd SQL command (enclosed between square brackets) on the **target system** immediately before reading the target table.
post={@sqlfile}|{[sqlcmd]}	odb will run a **single instance** of either sqlfile script or sqlcmd SQL command (enclosed between square brackets) on the **target system** immediately after the target table has been compared.

Diff operator	Meaning
tpar=num	Number of tables to compare in parallel when you have a list of tables in input
loaders=num	odb will use 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 8 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	This option will limit 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 won't find rows with differences in "non-key" columns
time	odb will print a "timeline" (milliseconds from start)

6 Using odb as a Query Driver (Technology Preview)

6.1 Getting CSV Output

It's often handy, while running performance tests to get a CSV output ready to be imported into your spreadsheet. You can easily get this kind of output with -c odb option. Let's analyze the following command:

```
$ ./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 will run:

- 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 4 threads: -T 4
- Omitting query text and query results (-q is equivalent to –q all): -q
- Printing a CSV output: -c

Producing 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
```

CSV output columns have the following meaning:

Column	Meaning
Thread ID	Thread ID. As we limited the number of threads to 4, thread id values are 0, 1, 2, 3
Proc ID	Execution number. As we have 11 executions in the 0-10 range
Thread Exec#	This is the progressive number (starting from 0) of execution for a specific thread
Script Cmd#	In case your script contains multiple SQL statement they will be numbered starting from zero
File	This is the script file name or "(null)" for –x commands
Label	This is the label assigned though "set qlabel" in the scripts
Command	First 30 characters of the SQL command. It will end with ">" if command text was truncated
Rows	The number of returned rows (not printed if you used –q)
Rsds	This is the Record Set Display Size. It 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

Column	Meaning
Stimeline	Queries start timeline in milliseconds
Etimeline	Queries end timeline in milliseconds

6.2 Assigning a Label to a Query

Sometimes it's not easy to recognize a query by reading the first 30 characters. odb lets you to assign a label to a generic query using `set qlabel <label>`. Have a look to the following script:

```
felici ~/Devel/odb $ cat script.sql
-- Trafodion 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,
...
```

When you run this script you will have the Query Label in the CSV output:

```
felici ~/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
```

6.3 Running 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 3 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 3 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:`)		
	Thread 1	**Thread 2**	**Thread 3**
nth execution	…	…	
4th execution	Script4.sql	Script4.sql	…
3rd execution	Script3.sql	Script3.sql	Script3.sql
2nd execution	Script2.sql	Script2.sql	Script2.sql
1st execution	Script1.sql	Script1.sql	Script1.sql

Round-Robin Assignment (es. `-P -3:`)		
Thread1	**Thread 2**	**Thread3**
…	…	
Script2.sql	Script3.sql	…
Script3.sql	Script4.sql	Script1.sql
Script4.sql	Script1.sql	Script2.sql
Script1.sql	Script2.sql	Script3.sql

6.4 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`	will execute 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`	will execute three times (`3:`) all files in the test/queries directory in parallel (`-P`), using five threads (`-T 5`) and in random order (`-Z`)

6.5 Defining a Timeout

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

Example:

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

It will execute, **serially,** all scripts with extension .sql under /home/mauro/scripts/; if the execution - after two hours (7200 seconds) is not completed, odb will stop.

6.6 Simulating User Thinking Time

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

Example:

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

It will run five times script1.sql using 2 threads. Each thread will wait 75 milliseconds before starting the next execution within a thread. You can also use a ***random thinking time*** in a given min:max range. For example the following command will start commands within a thread with a random delay between 50 and 500 milliseconds:

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

6.7 Starting Threads Gracefully

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

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

It will run five times script1.sql using 2 threads. Each thread will be started 200 milliseconds after the other.

> ***Note:*** *-delay introduces a delay during threads start-up while –ttime introduce a delay between one command and another within the same thread.*

6.8 Re-looping a Given Workload

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

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

will re-loop 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.

7 Using 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 here above) is used to specify the **.odbrc** section containing commands to be automatically executed when odb starts (see Running Commands When the Interpreter Starts).

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. So 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 the online 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 Customizing the Interpreter Prompt).

4. It's possible to run multi-threaded odb instances from within the single-threaded Interpreter with the ***odb*** keyword. This will run 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. It's possible to ***define aliases***. Example:

   ```
   root@MFDB[MFDB]SQL> set alias count "select row count from &1;"
   ```

   then, 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. Few 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#>|SMALL POLISHED>|        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.

This is the current odb SQL Interpreter help on line:

```
mauro pglocal[PUBLIC] (06:51:20) SQL> help
mauro pglocal[public] (16:57:52) 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;...] (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 #
```

7.1 Running 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 will:

- first check the ODB_INI environment variable

If this variable is not set, odb will

- look 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;
```

odb SQL Interpreter will automatically run 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.

7.2 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) "init file". `set prompt` recognize and expand the following variables:

1. **%U** → User name
2. **%D** →Data Source name
3. **%S** → Schema name
4. **%T** → Current Time
5. **%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)

So, for example:

```
set prompt "Prompt for %U connected via %D to %S in %M mode > "
```

will generate a prompt like this:

```
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. For 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 and the meaning of the ANSI color codes are:
   ^[[01;32m means green
   ^[[01;34m means blue
   ^[[00m means reset.

This is how my standard prompt looks:

A. Warnings, Limits, and Troubleshooting

1. odb uses Condition Variables to synchronize threads during copy and parallel load operations.
2. Most of the memory allocation operations are dynamic. So, for example, you can execute an SQL command as long as you want (in previous version of odb maximum SQL command length was 8192 bytes). However, I found it convenient to "hard code" the following limits :

```
#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 insane 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 check your default stack size:

```
s19user@traf64-dl2a ~/mf $ 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 << this is the problem
cpu time               (seconds, -t) unlimited
max user processes              (-u) 2047
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

and reset it to sane values (value will be reset to the initial value when you start a new session). For example:

```
s19user@traf64-dl2a ~/mf $ ulimit -s 4096
```

4. If you get errors like this:

```
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 backend database doesn't use catalogs you should use an **empty string** as catalog name. Some buggy 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>`;
   - and 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)

B. How odb Is Coded and Tested

B.1 How odb Is Coded

For Trafodion Release 1.3, 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

B.2 How odb Is Tested

For Trafodion Release 1.3, the info, load, extract, and copy operations of odb have been fully tested.

Previously, odb had been tested using a set of 137 standard tests to check functionalities and identify memory/thread issues.

Nevertheless, there could be bugs somewhere.