SQLTEX v3.0

Oscar van Eijk

Sep 20, 2024

Contents

1 Introduction

SQLTEX is a preprocessor to enable the use of SQL statements in LATEX. It is a perl script that reads an input file containing the SQL commands, and writes a LATEX file that can be processed with your LATEX package.

The SQL commands will be replaced by their values. It's possible to select a single field for substitution substitution in your LATEX document, or to be used as input in another SQL command.

When an SQL command returns multiple fields and or rows, the values can only be used for substitution in the document.

1.1 Known limitations

- The LATEX `\includeonly` directive is ignored; all documents included with `\include` will be parsed and written to the output file.
- Currently, only 9 command- line parameters (1-9), and 10 variables (0-9) can be used in SQL statements.
- Replace files can hold only 1,000 items.

2 Installing SQLTEX

Since v3.0, SQLTEX is part of TEX Live and doesn't need further installation. If you are using a different LaTeX distro, please follow the steps below for your OS.

Before installing SQLTEX, you need to have it. The latest version can always be found at `https://github.com/oveas/sqltex`. The download consists of this documentation, an installation script for Unix (`install`), and the Perl script `sqltex`, and a replace- file (`SQLTeX_r.dat`) for manual installation on non- unix platforms[1].

---

[1] on Unix, this file will be generated by the install script

2.1 Requirements

SQLTeX requires the following software:

- Perl v5.10 or higher (`http://perl.org/`)
- Perl-DBI (`http://dbi.perl.org/`)
- The DBI driver for your database
  (see: `http://search.cpan.org/search?query=DBD%3A%3A&mode=module`)
- Getopt::Long (`https://metacpan.org/pod/Getopt::Long`)
- Term::ReadKey (`https://metacpan.org/pod/Term::ReadKey`)

2.2 Installation

If you are using a TeX Live distribution, SQLTeX is already available. For all other distros, follow the steps in this section.

First unpack the archive in a location of your choice and continue with one if the subsections below depending on you operating system.

2.2.1 Linux

Go to the top directory where the archive was unpacked (‘`cd sqltex-3.0`’) and execute the following commands:

```
$ ./configure [options]
$ make
$ [sudo] make install
```

In the last command, `sudo` is only required if the install destination (`PREFIX`, see below) is outside the own user environment.

For `configure`, the following options are user buy SQLTeX (type `./configure --help` for a full list):

`--prefix=PREFIX` install architecture-independent files in PREFIX. Default is `/usr/local`.

`--exec-prefix=EPREFIX` install architecture-dependent files in EPREFIX. Default is `PREFIX`.

The directives above are used by the ones below:

`--bindir=DIR` Location of the SQLTeX script. Default is `EPREFIX/bin`

`--sysconfdir=DIR` Location of the Configuration- and replacefiles. Default is `PREFIX/etc`

`--datarootdir=DIR` Data root, used by the directives below. Default is `PREFIX/share`

`--mandir=DIR` Location of the SQLTEX manpage. Default is `DATAROOTDIR/man`

`--docdir=DIR` Documentation root, used by `pdfdir` below. Default is `DATAROOTDIR/doc/sqltex`

`--pdfdir=DIR` Location of SQLTEX.pdf. Default is `DOCDIR`

After installation, the archive and unpack- directory can be removed.

2.2.2 Windows

Note: Since v3.0, the binary `SQLTEX.EXE` for Windows is not included in the distribution anymore[2]

The files `sqltex-3.0\sqltex`, `sqltex-3.0\src\SQLTeX.cfg` and `sqltex-3.0\src\SQLTeX_r.dat` must be placed manually in the directory of your choice, all in the same direcrtory.

2.2.3 OpenVMS

On OpenVMS the files must be copied manually to the destination. All files must reside in the same location:

```
$ COPY [.SQLTEX-3_0.SRC]SQLTEX. SYS$SYSTEM:SQLTEX.PL
$ COPY [.SQLTEX-3_0.SRC]SQLTEX.CFG SYS$SYSTEM:
$ COPY [.SQLTEX-3_0.SRC]SQLTEX_R.DAT SYS$SYSTEM:
$ SET FILE/PROTECTION=(W:RE) SYS$SYSTEM:SQLTEX.PL
```

Next, define the command `SQLTEX` by setting a symbol, either in the `LOGIN.COM` for all users who need to execute this script, or in some group– or system wide login procedure, with the command:

```
$ SQLTEX :== "PERL SYS$SYSTEM:SQLTEX.PL"
```

2.3 Configuration

The configuration file `SQLTeX.cfg` is located in `/usr/local/etc` (linux) or the same location where SQLTEX is installed (all other operating systems and in TEX Live distros)[3]. Multiple configuration files can be created, the command line option `--configfile` can be used to select the requested configuration.

Note: Use of the `--configfile` commandfile option can be disabled on system wide installations. To do so, the script `sqltex` must be modified.

---

[2] It can be generated with any (portable) perl version for Windows, like Strawberry Perl (`https://strawberryperl.com/`), with `PAR::Packer` (`https://metacpan.org/pod/PAR::Packer`) using the command:

```
pp -o sqltex.exe sqltex
```

[3] If a 1.x version of SQLTEX is installed on your system, make sure you save the configuration section, which was inline in older versions

At the top of the file (line 4), set the value for `$main::ext_cfgfile_allowed` to `0`.

Some values can be overwritten using command line options (see section 4.2). When the command line options are omitted, the values from the requested configuration file will be used.

dbdriver Database driver. The default is `mysql`. Other supported databases are `Pg`, `Sybase`, `Oracle`[4], `Ingres`, `mSQL`, `PostgreSQL` and `ODBC`[5], but also others might work without modification.

If your database driver is not support, look for the function `db_connect` to add support (and please notify me :)

oracle_sid Oracle Site Identifier, required when the `Oracle` database driver is selected.

odbc_driver Specification of the ODBC driver. Default is "`SQL Server`"

texex The default file extension for LaTeX file. When SQLTeX is called, the first parameter should be the name of the input file. If this filename has no extension, SQLTeX looks for one with the default extension.

stx An output file can be given explicitly using the '`--output`' option. When omitted, SQLTeX composes an output file name using this string.
E.g, if your input file is called `db-doc.tex`, SQLTeX will produce an outputfile with the name `db-docstx.tex`.

def_out_is_in By default, when no output file is specified or an output file without (relative) path is given, the output file will be generated in the current directory.
This behaviour changed in version 2.1. In older version, the location of the output file always was the same as the input file location. To revert to the old behaviour, set `def_out_is_in` to '`1`'[6].

multi_rfile If the commandline option `--replacementfile` is given, by default the given replacement file will be parsed and after that the default replacement file will be parsed as well.
If only the given replacement file should be parsed skipping the default file, set this value to `0`.

rfile_comment The comment-sign used in replace files. If this is empty, comments are not allowed in the replace files.

---

[4] This requires the configuration setting `oracle_sid`

[5] The actual driver can specified with the configuration setting `odbc_driver`

[6] Note the pre-v2.1 implementation also contained a bug: if the output file name contained an absolute or relative path, this path was always taken as relative from the input file location. In the new implementation, `def_out_is_in` is ignored if the output file name contains a path.

rfile_regexploc This must be part of the value `rfile_regexp` below.

rfile_regexp Explains how a regular expression is identified in the replace files (see section 2.4.1).

cmd_prefix SQLTEX looks for SQL commands in the input file. Commands are specified in the same way all LATEX commands are specified: a backslash (\) followed by the name of the command.
All SQLTEX commands start with the same string. By default, this is the string `sql`. When user commands are defined that start with the same string, this can be changed here to prevent conflicts.

sql_open This string is appended to the `cmd_prefix` to get the complete SQLTEX command for opening a database.
With the default configuration this command is "`\sqldb`".

sql_field This string is appended to the `cmd_prefix` to get the complete SQLTEX command to read a single field from the database.
With the default configuration this command is "`\sqlfield`".

sql_row This string is appended to the `cmd_prefix` to get the complete SQLTEX command to read one or more rows from the database.
With the default configuration this command is "`\sqlrow`".

sql_params This string is appended to the `cmd_prefix` to get the complete SQLTEX command to retrieve a list if fields that will be used as parameters (`$PAR1`, see section 4.1) in the multidocument environment (see section 3.7).
With the default configuration this command is "`\sqlparams`".

sql_update This string is appended to the `cmd_prefix` to get the complete SQLTEX command to update one or more rows in the database.
With the default configuration this command is "`\sqlupdate`".

sql_start This string is appended to the `cmd_prefix` to get the complete SQLTEX command start a section that will be repeated for every row from an array (see section 3.5).
With the default configuration this command is "`\sqlstart`".

sql_use This string is appended to the `cmd_prefix` to get the complete SQLTEX command use a named variable from the array that is currently being processed in a loop context (see section 3.5).
With the default configuration this command is "`\sqluse`".

sql_end This string is appended to the `cmd_prefix` to get the complete SQLTEX command to end a loop context (see section 3.5).
With the default configuration this command is "`\sqlend`".

sqlsystem_allowed Set this to "1" to allow the use of the `\sqlsystem` command (see section 3.6).

repl_step Replacing strings (see section 2.4 below) is done two steps, to prevent values from being replaced twice. This setting—followed by a three-digit integer - “000” to “999”—is used in the first step and replaces values from the first column. In the second step, values from the second column replace the temporary value.

If the first column in the replace file contains a character sequence that occurs in this temporary value, or if query results might contain the full string followed by three digits, this value might need to be changed in something unique.

alt_cmd_prefix In loop context, this setting is used internally to differentiate between sql statements to process immediately and sql statements on stack.

Normally, this setting should never change, but if the value for `cmd_prefix` has been changed and a conflict is found, the message “`Configuration item 'alt_cmd_prefix' cannot start with` *`<conflicting value>`*” indicates this setting should change as well.

2.4 Create replace files

Replace files can be used to substitute values in the output of your SQL commands with a different value. This is especially useful when the database contains characters that are special characters in LaTeX, like the percent sign (‘%’), underscore (‘_’) etc.

When SQLTeX is installed, it comes with a standard file—`SQLTeX_r.dat`—which is located in `/usr/local/etc`[7] (linux) or the same location where SQLTeX is installed (all other operating systems and in TeX Live distros).

Example:

```
$        \$
_        \_
%        \%
&        \&
<        \texttt{<}
>        \texttt{>}
{        \{
}        \}
#        \#
~        \~{}
\        \ensuremath{\backslash}
```

These are all single character replacements, but you can add your own replacements that consist of a single character or a character sequence (or even regular expressions, see section 2.4.1).

---

[7] if a replace file with that name already exists, it will be stored there as `SQLTeX_r.dat.new`

To do so, enter a new line with the character(string) that should be replaced, followed by one or more `TAB`-character(s) (*not* blanks!) and the character(string) it should be replaced with.
That last one can be empty if the input character(string) should be ignored, but the `TAB` after the input character(string) is mandatory!

If the first non-blank character is a semicolon (';'), the line is considered a comment line[8]. Blank lines are ignored.

The contents of the file are case sensitive, so of you add the line:

```
LaTeX         \LaTeX\
```

the word "LaTeX" will be changed, but "latex" is untouched.

Different replace files can be created. To select a different replace file for a certain SQLTeX source, use the commandline option '`--replacementfile` *`filename`*'. To disable the use of replace files, use '`no-replacementfile`'.

2.4.1 Regular expressions

The replace file can include regular expressions, which are recognized by a pattern given in the configuration setting `rfile_regexp`. A part of the pattern, configurable as `rfile_regexploc`, will be the actual regular expression.

By default, `rfile_regexploc` is "`...`" and `rfile_regexp` is "`re(...)`". If the sequence of three dots can appear anywhere else in the replace file, `rfile_regexploc` can be changed to any other sequence of characters, e.g. "`regexpHere`".
This also requires `rfile_regexp` to be changed. Its new value has to be "`re(regexpHere)`"

Both in the default configuration and with the modification example given above, the key for regular expressions is `re(<`*`regular expression`*`>)`, e.g.:

```
re(<p.*?>) \paragraph*{}
```

will replace all HTML <p> variants (<p style='font-size: normal'>, <p align='center'> etc)

An example replacement file using regular expressions to handle HTML codes could look like this:

```
&amp;            \&
<strong>         \textbf{
</strong>        }
<em>             \textit{
</em>            }
re(<br.*?/?>)    \\
re(<p.*?>)       \paragraph*{}
</p>             \\[0pt]
```

---

[8] in the default configuration. See the description for `rfile_comment` in section 2.3 to change of disable comment lines.

```
<sup>           $^{
</sup>          }$
re(<span.*?>)   \textsl{
</span>         }
re(<h1.*?>)     \section{
re(<h2.*?>)     \subsection{
re(<h3.*?>)     \subsubsection{
re(</h\d>)      }
```

3 Write your SQLTEX file

For SQLTEX, you write your LATEX document just as you're used to. SQLTEX provides you with some extra commands that you can include in your file.
The basic format[9] of an SQLTEX command is:
`\sql`*`cmd`*`[options]{SQL statement}`

All SQLTEX commands can be specified anywhere in a line, and can span multiple lines. When SQLTEX executes, the commands are read, executed, and their results—if they return any—are written to the output:

Input file:	*Output file:*
`\documentclass[article]`	`\documentclass[article]`
`\pagestyle{empty}`	`\pagestyle{empty}`
`\sqldb[oscar]{mydb}`	
`\begin{document}`	`\begin{document}`

Above you see the SQLTEX command `\sqldb` was removed. Only the command was removed, not the *newline* character at the end of the line, so an empty line will be printed instead. The example below shows the output if an SQLTEX command was found on a line with other LATEX directives:

Input file:	*Output file:*
`\documentclass[article]`	`\documentclass[article]`
`\pagestyle{empty}\sqldb[oscar]{mydb}`	`\pagestyle{empty}`
`\begin{document}`	`\begin{document}`

In these examples the SQLTEX commands did not return a value. When commands actually read from the database, the returned value is written instead:

---

[9] in this document, in all examples will be assumed the default values in the configuration section as described in section 2.3, have not been changed

Input file:	*Output file:*

`This invoice has \sqlfield{SELECT`
`COUNT(*) FROM INVOICE_LINE`
`WHERE INVOICE_NR = 20190062} lines.`
	`This invoice has 3 lines`

3.1 SQL statements

This document assumes the reader is familiar with SQL commands. This section only tells something about implementing them in SQLTeX files, especially with the use of command parameters and variables. Details about the SQLTeX commands will be described in the next sections.

Let's look at a simple example. Suppose we want to retrieve all header information from the database for a specific invoice. The SQL statement could look something like this:

```
SELECT * FROM INVOICE WHERE NR = 20190062;
```

To implement this statement in an SQLTeX file, the `\sqlrow` command should be used (see section 3.4):

First, it is important to know that SQL statements should *not* contain the ending semicolon (;) in any of the SQLTeX commands. The command in SQLTeX would be:

```
\sqlrow{SELECT * FROM INVOICE WHERE NR = 20190062}
```

Next, SQLTeX would be useless if you have to change your input file every time you want to generate the same document for another invoice.

Therefore, you parameters or variables can be used in your SQL statement. Parameters are given at the command line (see section 4.1), variables can be defined using the `\sqlfield` command as described in section 3.3.1.

Given the example above, the invoice number can be passed as a parameter by rewriting the command as:

```
\sqlrow{SELECT * FROM INVOICE WHERE NR = $PAR1}
```

or as a variable with the code line:

```
\sqlrow{SELECT * FROM INVOICE WHERE NR = $VAR0}
```

Note you have to know what datatype is expected by your database. In the example here the datatype is INTEGER. If the field "INVOICE_NR" contains a VARCHAR type, the `$PAR`ameter or `$VAR`iable should be enclosed by quotes:

```
\sqlrow{SELECT * FROM INVOICE WHERE NR = '$PAR1'}
```

3.2 Opening the database

Before any information can be read from a database, this database should be opened. This is done with the `\sqldb` command. `\sqldb` requires the name of the dabatase. Optionally, a username, password and remote database host can be given.

The format of the command is:

```
\sqldb[user=username,passwd=password,host=host]{database}
```

The command can be used anywhere in your input file, but should occur before the first command that tries to read data from the database.

If the keywords `user`, `passwd` and `host` are omitted, SQLTEX assumes the options are given in this order:
`\sqldb[username,password,host]{database}`
Default host is localhost, the default user is the current user.

Note: The `\sqldb` command cannot span multiple lines!

3.2.1 Prompt for password and/or username

If a password is omitted, SQLTEX will try connect to the database without a password, unless the commandline option `--password` is given (see section 4.2).

Forcing a user to enter a database password when SQLTEX runs can be achieved by specifying `?` as password:
`\sqldb[user=dbUser,passwd=?]{database}`

When different database users should be able to use the same SQLTEX file, the username can also be a question mark, forcing SQLTEXto prompt for a username:
`\sqldb[user=?,passwd=?]{database}`

3.3 Reading a single field

When a single field of information is to be read from the database, the command `\sqlfield` is used. By default, the command in the input file is replaced by its result in the output file.
The SQL command is enclosed by curly braces. Square brackets can optionally be used to enter some extra options. Currently, the only supported option is `setvar` (see section 3.3.1).
The full syntax or the `\sqlfield` command is:
`\sqlfield[options]{SELECT fieldname FROM tablename WHERE your where-clause}`
By default, the SQLTEX command is replaced with the value returned by the SQL query. This behaviour can be changed with options.

3.3.1 Define variables

The `\sqlfield` can also be used to set a variable. The value returned by the SQL query is not displayed in this case. Instead, a variable is created which can be used in any other SQL query later in the document (see also section 3.1).
Therefore, the option `[setvar=n]` is used, where n is an integer between 0 and 9.

Suppose you have an invoice in LATEX. SQLTEX is executed to retrieve the invoice header information from the database for a specific customer. Next, the invoice lines are read from the database.

You could pass the invoice number as a parameter to SQLTEX for use in your queries, but that could change every month. It is easier to :

- pass the customer number as a parameter,
- retrieve the current date (assuming that is the invoice date as stored in the database by another program), and store it in a variable:
  ```
  \sqlfield[setvar=0]{SELECT DATE_FORMAT(NOW(), "%Y-%m-%d")}
  ```
  This creates a variable that can be used as `$VAR0`,
- retrieve the invoice number using the customer number (a command line parameter, see also section 4.1) and the variable containing the invoice date. Store this invoice number in `$VAR1`:
  ```
  \sqlfield[setvar=1]{SELECT NR FROM INVOICES
  WHERE CUST_NR = ’$PAR1’ AND INVOICE_DATE = ’$VAR0’}
  ```
- use `$VAR1` to retrieve all invoice information.

The SQL queries used here do not display any output in your LATEX document.

3.4 Reading rows of data

When an SQL query returns more information than one single field, the SQLTEX command `\sqlrow` should be used. As with the `\sqlfield`, command, SQLTEX replaces the command with the values it returns, but `\sqlrow` accepts different options for formatting the output.

By default, fields are separated by a comma and a blank (‘`, `’), and rows by a newline character (‘\\’). To change this, the options “`fldsep`” and “`rowsep`” can be used.

e.g. In a `tabular` environment the fields should be separated by an ampersand (&), perhaps a line should separate the rows of information. (\\ `\hline`). To do this, the options can be used with `\sqlrow` as shown here:

```
\sqlrow[fldsep=&,rowsep=\\ \hline]{SELECT I.NR, A.NR, A.PRICE, I.AMOUNT,
(A.PRICE * I.AMOUNT) FROM ARTICLE A, INVOICE_LINE I WHERE I.NR = $VAR1
AND I.ARTICLE_NR = A.NR}
```

This will produce an output like:

```
1 & 9712 & 12 & 1 & 12 \\ \hline 2 & 4768 & 9.75 & 3 & 29.25 \\ \hline
3 & 4363 & 1.95 & 10 & 19.5 \\ \hline 4 & 8375 & 12.5 & 2 & 25 \\ \hline
```

3.4.1 Output rows on separate lines

Some LATEX packages require input on a separate line. If this output is to be read from a database, this can be set with the `rowsep` option using the fixed text “`NEWLINE`”.

Changing the example from section 3.4 above to:

```
\sqlrow[fldsep=&,rowsep=\\ \hline NEWLINE]{SELECT I.NR, A.NR, A.PRICE,
I.AMOUNT, (A.PRICE * I.AMOUNT) FROM ARTICLE A, INVOICE_LINE I WHERE
I.NR = $VAR1 AND I.ARTICLE_NR = A.NR}
```

would produce the following result:

```
1 & 9712 & 12 & 1 & 12 \\ \hline
2 & 4768 & 9.75 & 3 & 29.25 \\ \hline
3 & 4363 & 1.95 & 10 & 19.5 \\ \hline
4 & 8375 & 12.5 & 2 & 25 \\ \hline
```

3.4.2 Store data in an array

The `\sqlrow` command can also be used to store the data in an array. The value returned by the SQL query is not displayed in this case. Instead, an array is created which can be used later in the document in a loop context (see section 3.5).

Therefore, the option `[setarr=`*n*`]` is used, where n is an integer between 0 and 9.

3.5 Loop context

In a loop context, an array is filled with data from the database using `\sqlrow`. Later in the document, the data can be used in a text block that will be written to the output file once for every record retrieved.

The text block is between the `\sqlstart{`*n*`}` and `\sqlend{`*n*`}` commands, where n is the sequence number of the array to use[10].

Multiple text blocks can occur in the document, but they can *not* be nested!

In the example below, data for unpaid invoices is stored in an array identified with sequence number 0:

```
\sqlrow[setarr=0]{SELECT I.NR AS nr
      , I.DUE_DATE AS date
      , I.TOTAL AS amount
      , C.NAME AS customer
      FROM INVOICE I
      LEFT OUTER JOIN CUSTOMER C
         ON C.NR = I.CUST_NR
      WHERE I.PAY_DATE IS NULL}
```

To use this data, a text block must start with: `\sqlstart{0}`

Between this command and the first occurrence of `\sqlend{}`, an unlimited amount[11] of LaTeX text can be written. Within this text, every occurrence

---

[10] in `\sqlend`, the sequence number is ignored, but required by syntax.

[11] limited by your computer's memory only

of `\sqluse{<field name>}` will be replaced with the matching field from the current row, e.g.:

```
\sqlstart{0}
\begin{flushright}
Regarding: invoicenumber \sqluse{nr}
\end{flushright}

Dear \sqluse{customer},

On \today, the invoice with number \sqluse{nr}, payable before
\sqluse{date}, was not yet received by us.

We kindly request you to pay the amount of \texteuro\sqluse{amount}
as soon as possible.

\newpage
\sqlend{}
```

3.5.1 If-endif control block

In the loop context, parts of the document can be enabled if certain conditions are met, using a control block with `\sqlif{condition(s)}` and `\sqlendif{}`.

Conditions can be up to 2 conditions separated by an *and* (`&&`) or *or* (`||`).
The condition(s) consist of an left value and an right value seperated by 1 of the following comparisson operators: '`==`', '`!=`', '`<`',. '`>`', '`<=`' or '`>=`'. Numeric values will be used as is. When the value is text, it is expected to be the name of a field and '`\sqluse{}`' will be called to retrieve the value.

Example:

```
\sqlif{article_nr == 123 && \stock < 5}
Stock is below threshold, please reorder.
\sqlendif{}
```

Note the conditions are very basic with the following limitations:

- A maximum of 2 conditions is supported per if-statement.
- Only numeric comparissons are supported.
- If-elsif blocks cannot be nested.

When checks are needed that are not supported by SQLTEX, a workaround can be implemented in the SQL code.

3.6 Get input from external programs

The `\sqlsystem` command can be used to call commands at the operating system or external scripts and use their output in the location where the command was given. Any command arguments can be given in the command line.

When used in a loop context (see section 3.5), `\sqluse` can also be used to provide data to the script. If command arguments must be given for database access, the following tags can be used:

`<SRV>`	Name of the database server.
`<USR>`	Username to connect to the database.
`<PWD>`	Password to connect to the database.
`<DB>`	Name of the database.

They will be replaced by the credentials for connecting to the database (see section 3.2).

Example:

```
\sqlsystem{./add_vat --usr <USR> --db <DB> --pwd <PWD> ↩
--inv \sqluse{invoice_nr}}
```

By default, use of this command is disallowed. To enable it, set the value of `sqlsystem_allowed` to “1” in the configuration file (see also section 2.3.

If the command is disabled, occurances of this command will be replaced by the fixed text “`use of the \sqlsystem command is disallowed in the configuration`”.

Note: The `\sqlsystem` command cannot span multiple lines!

3.7 Output multiple documents

A single input file can be created to generate more output files using the `--multidoc-numbered` or `--multidoc-named` commandline option.

The input document must contain the command `\sqlsetparams` without any options. The query that follows can return an unlimited number of rows:

```
\sqlsetparams{SELECT NR, CUST_NR FROM INVOICES WHERE REMINDERS = $PAR1}
```

By processing this command, SQLTEX builds a list with all values retrieved and processes the input file again for each row.
In those runs, the queries are executed as described in the previous sections, using the returned fields to replace `$MPAR`*n* placeholders, where n starts with 1 and represents the fields in the order as they have been retrieved:

```
\sqlrow{SELECT * FROM INVOICES WHERE NR = $MPAR1}
\sqlrow{SELECT * FROM CUSTOMER WHERE CUST_NR = $MPAR2}
```

The options `--multidoc-numbered` or `--multidoc-named` cannot be used together.

Without these options, a parameter can be given and a single output document will be created, ignoring the `\sqlsetparams` command.

With the `--multidoc-numbered` option, output filenames will be numbered *`filename`*`_1.tex` to *`filename`*`_`*`n`*`.tex`.
With the `--multidoc-named` option, output filenames will be numbered *`filename`*`_`*`parameter`*`.tex`, where *parameter* is the first value taken from the database (`$MPAR1`, the invoice number `nr` in the example above).
Note the parameter will not be formatted to be filename-friendly!

3.8 Update database records

Since version 1.5, SQLTEX supports database updates as well:
`\sqlupdate{UPDATE INVOICE SET REMINDERS = REMINDERS + 1, LAST_REMINDER = NOW() WHERE NR = $VAR1}`
This command accepts no options.

By default, the update statements will be ignored. To actually process them, the commandline options `--updates` must be given!

4 Process your SQLTEX file

To process your SQLTEX file and create a LATEX file with all information read from the database, call SQLTEX with the parameter(s) and (optional) commandline options as described here.

4.1 Parameters

SQLTEX accepts more than one parameter. The first parameter is required; this should be the input file, pointing to your LATEX document containing the SQLTEX commands.
By default, SQLTEX looks for a file with extension '`.tex`'.

All other parameters are used by the queries, if required. If an SQL query contains the string `$PAR`*`n`*[12], it is replaced by that parameter (see also section 3.1).

4.2 Command line options

SQLTEX accepts the following command- line options:

`--configfile` *`file`*, `-c` *`file`* SQLTEX configuration file. Default is `SQLTeX.cfg` in the systems default location (see section 2.3).

---

[12] where n is a number between 1 and 9. Note parameter '0' cannot be used, since that contains the filename!

`--file-extension` ***string***, `-E` ***string*** replace input file extension in outputfile: `input.tex` will be `input.`*`string`*.
For further notes, see option `--filename-extend` below.

`--filename-extend` ***string***, `-e` ***string*** add *string* to the output filename: `input.tex` will be `input`*`string`*`.tex`. This overwrites the configuration setting `stx`.
In *string*, the values between curly braces {} will be substituted:

- **P*n*** parameter *n*
- **M** current monthname (*Mon*)
- **W** current weekday (*Wdy*)
- **D** current date (*yyyymmdd*)
- **DT** current date and time (*yyyymmddhhmmss*)
- **T** current time (*hhmmss*)

e.g., the command

```
sqltex --filename-extend _{P1}_{W} my_file code
```

will read '`my_file.tex`' and write '`myfile_code_Tue.tex`'.
The same command, but with option `---file-extension` would create the outputfile `my_file._code_Tue`
The options `--file-extension` and `--filename-extend` cannot be used together or with `--output`.

`--force, -f` force overwrite of existing files. By default, SQLTEX exits with a warning message it the outputfile already exists.

`--help, -h` print this help message and exit.

`--multidoc-numbered, -m` Multidocument mode; create one document for each parameter that is retrieved from the database in the input document (see section 3.7). This option cannot be used with `--output`.

`--multidoc-named, -M` Same as `--multidoc-numbered`, but with the parameter in the filename instead of a serial number (see section 3.7).

`--null-allowed, -N` `NULL` return values allowed. By default SQLTEX exits if a query returns an empty set.

`--output` ***file***, `-o` ***file*** specify an output file. Cannot be used with `--file-extension`, `--filename-extend` or the `--multidoc` options.

`--skip-empty-lines, -S` All SQLTEX commands will be removed from the input line or replaced by the corresponding value. The rest of the input line is written to the output file. This includes lines that only contain a SQLTEX command (and a newline character). This will result in an empty line in the output file.
By specifying this option, these empty lines will be skipped. Lines that were empty in the input will be written.

`--write-comments, -C` LaTeX comments in the input file will be skipped by default. With this option, comments will also be copied to the output file.

`--prefix` *`prefix`*`, -p` *`prefix`* prefix used in the SQLTeX file. Default is `sql` (see also section 2.3 on page 6. This overwrites the configurarion setting `cmd_prefix`.

`--password` *`[password]`*`, -P` *[password]* database password. The value is optional; if omitted, SQLTeX will prompt for a password. This overwrites the password in the input file.

`--quiet, -q` run in quiet mode.

`--replacementfile` *`replace`*`, -r` *`replace`* Specify a file that contains the replace characters (see section 2.4).
Default is `SQLTeX_r.dat` in the systems default location (see section 2.4).
This default file will always be used after the given replacement file, unless `multi_rfile` is set to `0` in the configuration (see secion 2.3).

`--no-replacementfile, -R` Do not use a replace file. `--no-replacementfile` and `--replacementfile` *`file`* are handled in the same order as they appear on the command line, overwriting each other.
For backwards compatibility, `-rn` is also still supported.

`--sqlserver` *`server`*`, -s` *`server`* SQL server to connect to. Default is `localhost`.

`--updates, -u` if the input file contains updates, process them.

`--username` *`user`*`, -U` *`user`* database username. This overwrites the username in the input file.

`--version, -V` print version number and exit.

5 SQLTeX errors and warnings

`no input file specified`

SQLTeX was called without any parameters.
Action: Specify at least one parameter at the commandline. This parameter should be the name of your input file.

`File` *`input filename`* `does not exist`

The input file does not exist.
Action: Make sure the first parameter points to the input file.

`outputfile` *`output filename`* `already exists`

The outputfile cannot be created because it already exists.
Action: Specify another output filename with command line option `-e`, `-E` or `-o`, or force an overwrite with option `-f` (see also section4.2).

`no database opened at line` *line nr*

A query starts at line *line nr*, but at that point no database was opened yet.
Action: Add an `\sqldb` command prior to the first query statement.

`insufficient parameters to substitute variable on line` *line nr*

The query starting at line *line nr* uses a parameter in a WHERE- clause with `$PAR`*n*, where n is a number bigger than the number of parameters passed to SQLTeX.
Action: Specify all required parameters at the command line.

`trying to substitute with non existing on line` *line nr*

The query starting at line *line nr* requires a variable `$VAR`*n* in its WHERE- clause, where n points to a variable that has not (yet) been set.
Action: Change the number or set the variable prior to this statement.

`trying to overwrite an existing variable on line` *line nr*

At line *line nr*, a `\sqlfield` query tries to set a variable n using the option `[setvar=`*n*`]`, but `$VAR`*n* already exists at that point.
Action: Change the number.

`no result set found on line` *line nr*

The query starting at line *line nr* returned a `NULL` value. If the option `-N` was specified at the commandline, this is just a warning message. Otherwise, SQLTeX exits.
Action: None.

`result set too big on line` *line nr*

The query starting at line *line nr*, called with `\sqlfield` returned more than one field.
Action: Change your query or use `\sqlrow` instead.

`no parameters for multidocument found on line` *line nr*

SQLTeX is executed in multidocument mode, but the statement on line *line nr* did not provide any parameters for the documents.
Action: Check your query.

`too many fields returned in multidocument mode on` *line nr*

In multidocument mode, the lis of parameters retrieved on line *line nr* returned more than one fields per row.
Action: Check your query.

`start using a non-existing array on line` *line nr*

An `\sqlstart` command occurs, but refers to a non-existing array.
Action: Check the sequence number of the array filled with `\sqlrow[setarr=`*n*`]` and retrieved with `\sqlstart{`*n*`}` in your input file.

`\sqluse command encountered outside loop context on line` *`line nr`*

Data from array is used, but the current input file position is not in the context where this data is available.
Action: Check the presence and positions of the `\sqlstart` and `\sqlend` commands in your input file.

`unrecognized command on line` *`line nr`*

At line *line nr*, a command was found that starts with “`\sql`”, but this command was not recognized by SQLTEX˙
Action: Check for typos. If the command is a user- defined command, it will conflict with default SQLTEX commands. Change the SQLTEX command prefix (see section 2.3).

`no sql statements found in` *`input filename`*

SQLTEX did not find any valid SQLTEX commands.
Action: Check your input file.

6 Copyright and disclaimer

The SQLTEX project is available from GitHub: `https://github.com/oveas/sqltex`
For bugs, questions and comments, please use the issue tracker available at `https://github.com/oveas/sqltex/issues`

---

7 History

v3.0 *released: Sep 20, 2024*

- Made it possible to run SQLTEX directly from the distribution without `configure` and `make [install]` to make integration in TEX Live possible.
- Renamed the script to `sqltex`. For backwards compatibility, during installation on linux a symbolic link `SQLTeX` is created.
- The `SQLTeX.exe` binary is no longer included in the distribution (see footnote [2] on page 4).
- Removed support for the `--use-local-config` commandline option. The options `--configfile` and `--replacementfile` can be used instead.

- Added an option to disable the `--configfile` command line option (see note on page 4).
- Added the `--skip-empty-lines` and `--write-comments` command-line options.
- Added support for multiple replacement files.
- Fix: ordering in the replacement file.

v2.2 *released: Jul 31, 2024*

- Extended the default replace file (see 2.4) with more special characters (e.g. with diacritics) and HTML tags.
- Issue #6 (`https://github.com/oveas/sqltex/issues/6`): added support for ODBC drivers
- Issue #8 (`https://github.com/oveas/sqltex/issues/8`): added support for parameter-driven in `\sqlsetparams` statements (multi-document mode).
  Note: This requires an update of your input files for multi-document mode that have been created before v2.2. Refer to section 7.1.1 for details.
- Added the `\sqlsystem` command.
- Added the `\sqlif`-`\sqlendif` control block.

v2.1 *released: Jan 21, 2022*

- Fix bug #2 (`https://github.com/oveas/sqltex/issues/2`): standard path management for output files.
  See config item `def_out_is_in` in section 2.3 to revert to pre v2.1 behaviour.
- Fix: help was not displayed on Windows
- Implemented '?' as password in `dbopen`
- Implemented '?' as username in `dbopen`
- Implemented long options
- Allow overwriting variables in multidocument mode
- Added simple automated regression tests
- Added a man page for linux users
- Rewrote the installation procedure, now using `autotools` on linux.
- On linux, change the default installation directory to `/usr/bin` and store the configuration- and replacement files is `/etc`.
- Added option `--use-local-config`.

v2.0 *released: Jan 12, 2016*

- Fix: Oracle support using ORASID

- Fix: Ensure replacements are handled in the same order as they appear in the replacements file
- Separate configuration file(s)
- Added the options `-c` and `-M`
- Support for regular expressions in replace files
- Implemented support for the LaTeX `\input` and `\include` directives
- Implemented loop context
- Skip commentlines
- Project moved from local CVS to GitHub

v1.5 *released: Nov 23, 2007*

- Support for multiple databases
- Implemented database updates (`sqlupdate`)
- Implemented multiple output documents (option `-m`)

v1.4.1 *released: Feb 15, 2005*
Fix: removed leading whitespaces added to database results before replace

v1.4 *released: May 2, 2002*
Implemented replace files

v1.3 *released: Mar 16, 2001*
First public release

7.1 Changes that require updates in your input files

7.1.1 Multi-document mode since v2.2

Up until v2.1, the statement in `\sqlsetparams` could return only one field per row and the statement itself could not handle parameters. The placeholder `$PAR1` was reserved for the subsequent statements.

Since v2.2 it is possible to retrieve multiple values per row. They will replace the placeholders `$MPARn` in the subsequent statements, while `$PARn` placeholders can now also be used for regular parametes in the `\sqlsetparams` statement itself.

This means, in input documents created before v2.2, all “`$PAR1`” placeholders must be replaced by “`$MPAR1`”.