dbpre

precompiler for GnuCOBOL and MySQL

Version 0.4

Users Guide

August 2020

Table of content

License....3
Introduction....4
Requirements....6
Usage of dbpre....7
Compiling dbpre....8
Compiling the API cobmysqlapi....9
dbpre options....10
Fixed column format and free format....11
Suffixes for COBOL source code....12
The framework PGCTB....14
    The *.param file....16
The examples....17
Creating the MySQL database for the examples....18
Compiling under Windows....19

License

Introduction

dbpre is a precompiler for MySQL and GnuCOBOL, so a precompiler for SQL, which replaces the EXEC SQL statements in COBOL source code with the calls to MySQL to perform the wanted functions.

dbpre comes with

- dbpre – the precompiler itself
- cobmysqlapi – the API to MySQL
- PGCTB – a framework to make it easier to program programs with Data Base access (PGCTB = Pipers Gnu Cobol Tool Box)
- two examples
- this document and some other documents

The precompiler dbpre processes the input file with the suffix *.scb (SQL COBOL) and creates a *.cob file.

It goes like this:

The COBOL source code has the suffix *.scb (SQL COBOL) and may, but must not, include the framework PGCTB.

You can of course write your own SQL statements instead of using the framework, all up to you and your needs.

dbpre processes the *.scb file and generates a *.cob and *.lst file and reports errors, if some, related to SQL or missing copybooks and such, appear.

The by dbpre generated *.cob file can then be compiled with GnuCOBOL‘s cobc, linked together with cobmysqlapi.o, the API, which does the communication with MySQL.

Or even other COBOL compilers, but that wasn‘t tested right now.

And thats it, the result should be a working binary, which can access MySQL databases.

Requirements

GnuCobol installed

`https://sourceforge.net/projects/gnucobol/`

MySQL installed

```
sudo apt-get install mysql-client mysql-server
```

or

https://dev.mysql.com/doc/mysql-installation-excerpt/5.7/en/

gcc installed

(usually default for Linux systems)

Usage of dbpre

dbpre is a precompiler, it processes COBOL source code with SQL statements and creates an output, a COBOL source code, which then can be compiled with GnuCOBOL‘s cobc.

So, first run the precompiler dbpre to convert the EXEC SQL statements into API calls (cobmysqlapi), then use dbpre‘s output and compile it with GnuCOBOL‘s cobc, linking it with cobmysqlapi.o.

Or any other COBOL compiler, but that wasn‘t tested, but there might be a chance, that it works.

Compiling dbpre

Use the script „c“ to compile dbpre.c or type:

```
gcc dbpre.c -odbpre -O
sudo cp dbpre /usr/local/bin
```

The result should be the binary dbpre, the executable precompiler itself.

To test it, type **`dbpre`** or **`dbpre -h`** or **`dbpre -v`**, that should produce some output.

Compiling the API cobmysqlapi

This is the API, the layer between GnuCOBOL and MySQL.

It must be linked to the COBOL program, which wants to interact with MySQL, this is the connection from the EXEC SQL statements in the COBOL source code (*.scb) to MySQL.

To compile it, use the shell script „capi“, which will generate cobmysqlapi.o, which must be linked with the COBOL program.

Or type

```
gcc -I/usr/include/mysql -c cobmysqlapi.c
cp cobmysqlapi.o ../examples
```

Of course, MySQL and it‘s header files must be installed to compile the API.

dbpre options

```
$ dbpre -h
dbpre V 0.4 2020-08-22
dbpre [options] progname

Options
=======
-h, --help      - this message
-v, --version   - print version of dbpre
-I              - specify path for copybooks, e.g -I/tmp/copybooks/
-ts    - tab stop, tabs will be expanded to nnn spaces, e.g. -ts=3
-f / --free     - free format source code formatting allowed
```

The options are used to control what dbpre does or does not.

`-I` (= include) specifies one path to a directory where dbpre looks for copybooks which are specified with the `COPY` or `EXEC SQL INCLUDE` commands. Maybe in the future dbpre will support more than only one path.

`-ts` (= tab stop) If the COBOL source code contains tab characters dbpre replaces each tab character with a given amount of spaces. The default is 3 spaces for each tab character. Specify `-ts=8` and dbpre replaces each tab character with 8 spaces.
Most editors, like `vi`, insert tab characters into the source code when you press the TAB-key.

`-f` (= free format) dbpre by default expects COBOL source code in fixed column format. To tell dbpre that the source code is in free format (format without fixed colums for coding, comments and such), use this option.

Fixed column format and free format

COBOL in fixed column format looks like this:

```
----+-*--1----+----2----+----3----+----4----+----5----+----6----+----7-*--+----8
      **************************************************************************
      *  I D E N T I F I C A T I O N   D I V I S I O N                         *
      **************************************************************************
        IDENTIFICATION              DIVISION.
        PROGRAM-ID.                 PCTB003B.
        AUTHOR.                     THE_PIPER.
        DATE-WRITTEN.               TODAY.
      /
      **************************************************************************
      *  D A T A    D I V I S I O N                                            *
      **************************************************************************
        DATA                        DIVISION.
```

As you can see, an asterik in column 7 specifies a comment line, a slash „/“ in column 7 forces a page break, COBOL statements are written at column 9, and so on.

Free format has no limitations, like an asterik in column 7 or „/“ in column 7 forces a page break and such.

It looks like this:

```
*>*************************************************************************
*>  I D E N T I F I C A T I O N   D I V I S I O N                         *
*>*************************************************************************
IDENTIFICATION              DIVISION.
PROGRAM-ID.                 freeform.
AUTHOR.                     THE_PIPER.
DATE-WRITTEN.               TODAY.
*>*************************************************************************
*>  D A T A    D I V I S I O N                                            *
*>*************************************************************************
DATA                        DIVISION.
```

A comment starts with „*>“ at any position in a line, COBOL statements can start at any column in a line, in this example, they are written at column 1.

Suffixes for COBOL source code

The input for dbpre, the COBOL source code with EXEC SQL statements, must have the suffix „scb“, meaning SQL COBOL.

The output of dbpre has the suffix „cob“, meaning COBOL. This is the input for GnuCobol‘s cobc.

Example:

A SQL statement in a *.scb file looks like this:

```
EXEC SQL
   DELETE
   FROM example_table
END-EXEC.
EVALUATE TRUE
  WHEN DB-OK
     CONTINUE
  WHEN OTHER
     PERFORM DB-STATUS
END-EVALUATE
```

dbpre‘s result in the created *.cob file then looks like this:

```
DBPRE      MOVE 8             TO SQLCA-SEQUENCE
      *    EXEC SQL
DBPRE *    END-EXEC.
DBPRE   MOVE LOW-VALUES TO SQLCA-STATEMENT
DBPRE   STRING
DBPRE   'DELETE ' DELIMITED SIZE
DBPRE   'FROM ' DELIMITED SIZE
DBPRE   'example_table ' DELIMITED SIZE
DBPRE   INTO SQLCA-STATEMENT
DBPRE   END-STRING
DBPRE   CALL 'MySQL_query' USING SQLCA-STATEMENT
DBPRE   END-CALL
DBPRE   MOVE RETURN-CODE TO SQLCODE
           EVALUATE TRUE
             WHEN DB-OK
                CONTINUE
             WHEN OTHER
                PERFORM DB-STATUS
           END-EVALUATE
```

And this *.cob file can be compiled with cobc.

The framework PGCTB

PGCTB stands for Pipers Gnu Cobol Tool Box.

This is a framework which makes writing programs, which access MySQL, easier, because it does things every program like this needs for the user.

Like connecting to the database, opening it, error handling, rollbacks, commit the result, close the database and display banners at program start and termination.

You can use this framework, but dbpre is working too without it.

So it‘s up to you if you‘re using it or not.

First, the framework needs some definitions in the working storage, so include this copybook:

```
         WORKING-STORAGE SECTION.
      *
      * The needed working storage stuff for the framework
         COPY PGCTBBATWS.
```

Then, the coding of the framework itself in the PROCEDURE DIVISION

```
      *****************************************************************
      *  P R O C E D U R E   D I V I S I O N                          *
      *****************************************************************
         PROCEDURE DIVISION.
      * The framework itself, calling PGCTB-ACTION to run the users coding
            EXEC SQL
                INCLUDE PGCTBBAT REPLACING 'TTTTNNNB' BY 'PCTB003B'.
            END-EXEC.
```

Here you must replace the TTTTNNNB with the real name of the program.
This will be displayed in the banners at program start/termination.
The length of the program name in PGCTB is limited to 8 characters.

And, as written above, the framework calls a section named PGCTB-ACTION.

This is where the users coding goes, PGCTB does all that basic stuff like connecting to the database, opening it, then calls PGCTB-ACTION, and after that, closes the database, or, in case of errors, does a rollback.

```
      *****************************************************************
      *  P G C T B - A C T I O N   S E C T I O N                     *
      *****************************************************************
        PGCTB-ACTION SECTION.
      *
           DISPLAY 'In PGCTB-ACTION.'

           PERFORM DISPLAY-ALL-RECORDS
      *
           DISPLAY 'Delete entire table'
      *
           EXEC SQL
              DELETE
              FROM example_table
           END-EXEC.
           EVALUATE TRUE
             WHEN DB-OK
[OTHER STUFF]
      *
           PERFORM DISPLAY-ALL-RECORDS
      *
           DISPLAY 'Ende PGCTB-ACTION.'
      *
           .
        PGCTB-ACTION-EXIT.
           EXIT.
```

Important: PGCTB-ACTION must end with an EXIT, not with a STOP RUN.

STOP RUN is absolutely forbidden, when using this framework, further ROLLBACK and COMMIT.

Use the appropiated features of the framework, if a DB error occurs, PERFORM DB-STATUS, this will cause a ROLLBACK and display an error message.

If no error occurs, leave PGCTB-ACTION with EXIT and the framwork will do a COMMIT and closes the database.

Example of error handling after an EXEC SQL statement:

```
           EVALUATE TRUE
             WHEN DB-OK
                CONTINUE
             WHEN OTHER
                PERFORM DB-STATUS
           END-EVALUATE
```

The *.param file

The framework needs to know to connect to which DB, with which user and password and such.

These informations are stored in a *.param file, like PCTB003B.param

It looks like this:

```
DBHOST=localhost
DBUSER=root
DBPASSWD=YourPasswordHere!
DBNAME=testdb
DBPORT=03306
DBSOCKET=null
```

DBHOST – where is MySQL running, in this case not an URL or an IP address, right on this machine, localhost

DBUSER – usually root

DBPASSWD – set this to your actual password for DBUSER

DBPORT – the port MySQL uses, usually 3306

DBSOCKET – null, unused

The examples

There are two examples for dbpre, PCTB003B and freeform.

Both examples do the same job, but PCTB003B is written in fixed columns format, and freeform is written in free format.

To run those examples, you need to create a table in MySQL, see next chapter, how to do that.

Both programs do:

1. Delete all records in example table
2. Insert 10 records in example table
3. Display all records
4. Update record 1 and 3
5. Display all records

Thats it.

The files of the two examples are:

cPCTB003B	-shell script to preprocess and compile PCTB003B.scb
PCTB003B.scb	- the COBOL source code, input for dbpre
PCTB003B.param	- parameter file for the framework PGCTB
PCTB003B.sh	- shell script to run the compiled binary
cfreeform	-shell script to preprocess and compile freeform.scb
freeform.scb	- the COBOL source code, input for dbpre
freeform.param	- parameter file for the framework PGCTB
freeform.sh	- shell script to run the compiled binary

Creating the MySQL database for the examples

To use the sample programs, you have to create a table in your MySql database, which must look like this:

Name: example_table:

```
+--------+----------+------+-----+---------+-------+
| Field  | Type     | Null | Key | Default | Extra |
+--------+----------+------+-----+---------+-------+
| field1 | char(20) | NO   |     | NULL    |       |
| field2 | char(16) | NO   |     | NULL    |       |
| field3 | char(32) | NO   |     | NULL    |       |
+--------+----------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Example:

```
sudo mysql -p

create database testdb;

use testdb;

create table example_table(field1 char(20), field2
char(16), field3 char(32));
```

And you have created the needed table in MySQL for the examples.

Compiling under Windows

Install GnuCOBOL from here: https://arnoldtrembley.com/GnuCOBOL.htm#binaries

it includes too a gcc.exe for windows.

Use this gcc to compile dbpre.c and cobmysqlapi.c.

```
gcc -I/usr/include/mysql -c cobmysqlapi.c
```

The path after -I must point to the location of mysql.h, which depends on your installation of MySQL.

(This needs some more testing and more details...)