The eXtended subsystem: POSTGRESQL

This subsystem is intended to provide a native POSTGRESQL support to TF PROTOCOL. Be aware that the implementation of the server is not compelled to implement any of the extended subsystems. The client can test if this subsystem is implemented by using one of its commands. If the server responses “UNKNOWN”, you will realize that the subsystem is not implemented.

XS_POSTGRESQL

XS_POSTGRESQL makes the server enter the subsystem. Once inside the subsystem, the server remains there until the client explicitly exits the module with the proper commands. If this module is not implemented by the server, it will return UNKNOWN. Otherwise the eXtended Subsystem POSTGRESQL will start listening for commands after sending OK. The return status for this command could be:

OK
UNKNOWN

Every command below will transmit before the payload a header indicating the size of the payload that is coming next. This header is a 64bit signed integer converted to network byte-order (big-endian).

OPEN “host” “port” “user” “password” “dbname”

OPEN opens a database in the host indicated by the first parameter. The “port” parameter is the port in which the mysql server is listening. The “user” “password” parameters are used to authenticate

the user in the server database. The “dbname” parameter is used to identify the desired database among all others. The return status for this command could be:

1 BD OPENED OK WITH ID “ID”
2 FAILED OPEN POSTGRESQL CONNECTION

If the command succeeds the ID returned in “ID” -without the double quotes- will represent that particular opened database. From now on, any of the possible operations will require that ID to identify the database among all opened databases.

The DB-ID is a 64bit signed integer in its text representation. As this number may range from -(2^63)-1 up to 2^63-1, its text representation could be from 1 up to 20 bytes, every byte representing one decimal digit. 8 / 2 * 3 + 8. A buffer of 21 bytes should be enough to store all digits plus a possible sign (-).

EXEC “DB-ID” “SQL-QUERY”

EXEC executes an SQL-Query in the database represented by the specified handle in “DB-ID”. The parameters are separated by a whitespace and the double quotes must not be included. The return status for this command could be:

5 EXEC OK
6 FAILED EXEC BD DOES NOT EXISTS
7 FAILED EXEC POSTGRESQL

If EXEC returns OK -code 5- it means that the query has succeeded. May there be data to transmit or not. In any case when there is no more data to send, the header will have 0 has its value. This means

that after the client reads 5 EXEC OK, it must continue reading from the network until the header sent previously to every payload is 0.

If there is no data to send, immediately after the client receives 5 EXEC OK will receive a header with value 0. If there is data to send, after the 5 EXEC OK, the server will send the names of the fields separated by two @@ like in the example below:

Name@@Last-Name@@etc

After the client receives the fields names, the server will send every row that the executed query has returned, formatted like the above example. This is, by separating the fields values by two @@:

John@@McDonald@@etc

EXECOF “path/to/file” “DB-ID” “SQL-QUERY”

EXEC executes an SQL-Query in the database represented by the specified handle in “DB-ID”. The result of the query is stored in the file indicated in the first parameter. If the file does not exist, then is created; otherwise, the resulting data from the query will be appended to the file. The format used to store the data is the same used by EXEC. The parameters are separated by a whitespace and the double quotes must not be included. The return status for this command could be:

8 EXECOF OK
9 FAILED EXECOF BD DOES NOT EXISTS
10 FAILED EXECOF POSTGRESQL

CLOSE “DB-ID”

CLOSE closes the database represented by the handle “DB-ID”. This frees that handle to be reused in next openings. The return status of this command could be:

3 CLOSE OK
4 FAILED CLOSE BD DOES NOT EXISTS

EXIT

EXIT exits the XS_POSTGRESQL module without freeing the previously allocated resources. This is, every opened database will remain as such and the returned handles, valid. However, you are now at the main command interface of the TF PROTOCOL. If you want enter again the POSTGRESQL subsystem, you can use the XS_ POSTGRESQL command again.

TERMINATE

TERMINATE exits the XS_ POSTGRESQL module, unlike EXIT command, freeing the previously allocated resources. This is, every opened database will be closed and every handle unallocated. Now you are in the main command interface of the TF PROTOCOL. You may enter the subsystem again by using the XS_ POSTGRESQL command.