Subject: Quick explanation of function calls for postgresql
Posted by captainc on Mon, 30 Jun 2008 20:50:25 GMT

I've been looking over the examples for postgres as I would like to use the functionality, but I am confused as to what exactly some of the functions do. I have been browsing the source, but I think it would help if we had a quick explanation of the structure of the code and some of these functions.

from the example, this is the section that confuses me most:

```
Progress p;
p.SetText(t_("Creating _DEBUG database"));
SqlSchema sch(PGSQL);
StdStatementExecutor se(m_session);
All_Tables(sch);
if(sch.ScriptChanged(SqlSchema::UPGRADE))
PostgreSQLPerformScript(sch.Upgrade(),se, p);
if(sch.ScriptChanged(SqlSchema::ATTRIBUTES)) {
PostgreSQLPerformScript(sch.Attributes(),se, p);
}
if(sch.ScriptChanged(SqlSchema::CONFIG)) {
PostgreSQLPerformScript(sch.ConfigDrop(),se, p);
PostgreSQLPerformScript(sch.Config(),se, p);
}
sch.SaveNormal();
```

The functions/lines I am not sure about are
All_Tables() --> Load information from all the tables of the db?
ScriptChanged() --> Check if table info different from defined schema?
PostgreSQLPerformScript()
Attributes()
ConfigDrop()
Config()
SaveNormal() --> Save all changes back to database?

Subject: Re: Quick explanation of function calls for postgresql
Posted by mirek on Wed, 02 Jul 2008 11:01:20 GMT

captainc wrote on Mon, 30 June 2008 16:50I've been looking over the examples for postgres as I would like to use the functionality, but I am confused as to what exactly some of the functions do. I have been browsing the source, but I think it would help if we had a quick explanation of the structure of the code and some of these functions.

from the example, this is the section that confuses me most:

```
Progress p;
p.SetText(t_("Creating _DEBUG database"));
SqlSchema sch(PGSQL);
StdStatementExecutor se(m_session);
All_Tables(sch);
if(sch.ScriptChanged(SqlSchema::UPGRADE))
PostgreSQLPerformScript(sch.Upgrade(),se, p);
if(sch.ScriptChanged(SqlSchema::ATTRIBUTES)) {
PostgreSQLPerformScript(sch.Attributes(),se, p);
}
if(sch.ScriptChanged(SqlSchema::CONFIG)) {
PostgreSQLPerformScript(sch.ConfigDrop(),se, p);
PostgreSQLPerformScript(sch.Config(),se, p);
}
sch.SaveNormal();
```

All these are related to "uploading" .sch file into database.

SqlSchema is instance to hold uploading SQL scripts.

All_Tables will "dump" SQL scripts from .sch file and stores them into database. This function is defined by including .sch file. It is sort of .sch interface point.

There is a couple of "standard" scripts.

UPGRADE script contains creates all tables and columns, however does so by creating tables only with the first column, then adding all columns (using ALTER TABLE ... ADD COLUMN) one by one. This allows incemental development of the modle - at start of app, model gets upgraded, commands to create all columns and tables that already exist simply fail, but any new columns or tables are added. Note that this model does not support removing columns or changing datatype - that has to be done manually.

ATTRIBUTES script adds any "attributes", namely constraints or indicies. The main reason to have this separated is because sometimes contraints have to be added only after the tables are defined (forward foreign keys). Secondary reason is that U++ also generates "drop" scripts; sometimes it is useful, when maintaining app, to drop all or some contraints and indicies and recreate them later.

CONFIG contains any data configuration - usually inserts of "metadata". Frankly, it is less important and in many cases this can be easily done some other way.

SaveNormal saves all scripts into U++ config dir (.exe dir in Win32, ~/.upp/appname in Linux).

sch.ScriptChanged compares script stored in sch with script stored by SaveNormal in previous run of code. This is to avoid running scripts each time (it can be time consuming). OTOH, sometimes this incorrectly blocks script execution, for example, if you are for some reason using

two identical database schemas, you upload to one (thus in this session SaveNormal overwrites script files), then run app again with second schema - and files are identical. If I have this problem, I usually just delete scripts (they are not equal then .

Mirek

Subject: Re: Quick explanation of function calls for postgresql
Posted by captainc on Wed, 02 Jul 2008 14:08:15 GMT
View Forum Message <> Reply to Message

Thank you for the explanation; very helpful.
Quote:All_Tables will "dump" SQL scripts from .sch file and stores them into database.
If this is going to execute the scripts anyway, what is the use for checking for changes to update immediately after? Will All_Tables() only execute the "create tables" sql from the "S_*.sql" file, whereas Upgrade() will execute the "U_*.sql" file?
Also, is there an equivalent to the All_Tables() function that will only compare the current scripts with the potential new ones and not execute the create statements (even though the create statements simply fail when the table already exists...)?

Subject: Re: Quick explanation of function calls for postgresql
Posted by mirek on Wed, 02 Jul 2008 20:50:20 GMT
View Forum Message <> Reply to Message

captainc wrote on Wed, 02 July 2008 10:08Thank you for the explanation; very helpful.
Quote:All_Tables will "dump" SQL scripts from .sch file and stores them into database.
If this is going to execute the scripts anyway, what is the use for checking for changes to update immediately after? Will All_Tables() only execute the "create tables" sql from the "S_*.sql" file, whereas Upgrade() will execute the "U_*.sql" file?

All_Tables does not execute scripts, it creates them as Strings inside SqlSchema.

Checking for changes is useful mostly because executing scripts can take quite long and you do not want it to happen at each app start.

BTW, I usually create/upload scripts only in debug mode, while developing the app. Release mode has these parts excluded (by #ifdef).

Quote:
Also, is there an equivalent to the All_Tables() function that will only compare the current scripts with the potential new ones and not execute the create statements (even though the create statements simply fail when the table already exists...)?

I am not quite sure where are you aiming at.

Well, the much more advanced version of SqlSchema would perhaps tried to read the schema from DB and update only what is necessarry. That would also solve the problem of changing column types, removing columns or tables etc...

Anyway, it would also make it much much more complicated and dependant on specific RDBMS. Current model is not perfect, but is simple and works quite well.

Mirek