# U++ SQL

## Incremental Development of Database Models/Schemas

### *Creating the database:*

```
SqlSchema sch(PGSQL);
All_Tables(sch);
```

The `All_Tables` function will generate SQL scripts from your schema (.sch file) definition and store them internally. This function is defined by including the .sch file. It is a sort of .sch interface point. All_Tables does not execute scripts, it creates them as Strings inside SqlSchema.

### *Checking for changes and executing SQL creation/update/drop scripts:*

```
StdStatementExecutor se(m_session);
Progress p;
p.SetText(t_("Creating database"));
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);
}
```

The `ScriptChanged()` function compares the current existing SQL script stored by `SaveNormal` in previous run of code (by default in the .exe dir in Win32, and ~/.upp/appname in Linux) to the one generated by the application at runtime. If they are different, then `ScriptChanged()` will return true. The purpose is to avoid running scripts each time (it can be time consuming).

`ScriptChanged()` also takes as arguments the directory to look in for the scripts and the base-name of the scripts. Customizing the location of the script files is especially useful for connecting to multiple databases.

The UPGRADE script contains the SQL code to create all tables and columns, however it does so by creating tables only with their first column, then adding all columns using 'ALTER TABLE ... ADD COLUMN' one by one. This allows incemental development of the model. At the start of the application, the model gets upgraded, and commands to create all columns and tables that already exist simply fail, but any new columns or tables will be 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 (Ie. forward foreign keys). A 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".

Note: in the above code, `Progress p` is used as a callback function that updates the progress indicator (optional);

### *Saving your SQL script files to disk:*

```
sch.SaveNormal();
```

`SaveNormal` saves all scripts into a configuration directory (.exe dir in Win32, ~/.upp/appname in Linux). This function also takes as arguments the directory and base file name of the script files.

```
SaveNormal(const char *dir = NULL, const char *name = NULL)
```

Note: In some cases, script execution can be blocked in your program if you do not customize the directory location and/or the base-name of the script files. For example, say you are using 2 identical database schemas and you save to one database, saving the script files using `SaveNormal()`. Then, you attempt to run it against another database, the script files on disk are already created and can be the same, and they will not be recognized as changed by your program. If you only have 1 database and schema, this should not be an issue.