Databaskoppling från klientprogram

Access till PostgreSQL från C/C++

Alla större databashanterare erbjuder ett interface för att administrera databasen och ställa databasfrågor från högnivåspråk som Java, C/C++, Python m.fl. I Java t.ex. används JDBC (Java Database Connectivity) som ger ett standardiserat interface till olika databaser. I den här lektionen och dess tillhörande inlämningsuppgift ska vi använda API:et libpq från PostgreSQL för att accessa en databas inifrån ett C++-program. I korthet består libpq av en header-fil, libpq-fe.h, som definierar C-funktioner, konstanter med mera samt ett antal lib-filer som innehåller de kompilerade funktionerna. Header-filen ska inkluderas vid kompilering och lib-filerna ska tillföras vid länkningen av programmet. I lektionen tittar vi bara på ett mycket begränsat antal funktioner för att kunna ansluta till databasen, ställa SQL-frågor och hämta resultaten.

PQconnectdb

```
PGconn *PQconnectdb(const char *conninfo);
```

Denna funktion skapar en förbindelse till databasen, conninfo är en sträng med inloggningsinformation. Resultatet returneras som en pekare till PGconn struct som sedan skickas med som argument till andra PQ-funktioner.

Typiskt för pqlib är att resultatet ofta fås som en pekare till en struct som funktionen har allokerat dynamiskt. När den inte längre behövs åligger det klientkoden att deallokera minne och det görs genom anrop till PQclear(pekare till struct).

PQstatus

```
ConnStatusType PQstatus(const PGconn *conn);
```

Denna funktion kontrollerar status för en förbindelse och returnerar resultatet i en ConnStatusType som kan vara CONNECTION_OK eller CONNECTION_BAD. Vid feltillstånd, t.ex. efter CONNECTION_BAD kan ett felmeddelande fås genom PQerrorMessage(PGconn *) som returnerar en C-sträng.

PQexec

```
PGresult *PQexec(PGconn *conn, const char *query);
```

Med denna funktion kan vi exekvera SQL-frågor. Returvärdet är antingen en pekare till resultatet eller en nullptr.

En SQL-fråga mot en databas som har flera samtidiga användare görs som en transaktion för att resultatet inte ska kunna påverkas av andra parallella accesser till samma tabeller. Före varje fråga gör vi därför ett anrop av typen:

```
PGresult *res = PQexec(conn, "BEGIN");
```

och när vi är färdiga med resultatet anropar vi

```
PGresult *res = PQexec(conn, "END");
```

Dessa båda anrop ramar in ett transaktionsblock där vi gör våra SQL-frågor.

Om PGresult-pekaren inte är en nullptr så kan vi via pekaren hämta svaren.

```
int PQntuples(const PGresult *res);
```
Returnerar antal rader i svaret.

```
int PQnfields(const PGresult *res);
```
Returnerar antalet kolumner.

```
char *PQfname(const PGresult *res, int field_num);
```
Returnerar namnet på field_num kolumnen som en C-string.

```
char *PQgetvalue(const PGresult *res, int tup_num, int field_num);
```
Returnerar värdet på tup_num raden field_num kolumnen som en C-string.

Fullständig dokumentation för libpq finns på
https://www.postgresql.org/docs/current/libpq.html.

MD5

```
text MD5(string)
```

Denna funktion ingår inte i libpq utan är en inbyggd funktion i PostgreSQL. Funktionen beräknar MD5-hash av strängargumentet och returnerar resultatet i hexadecimalt format (returneras som datatypen text i PostgreSQL). Funktionen är bra att känna till när ni ska lösa inlämningsuppgiften.

Följande exempel visar har vi använder MD5-funktionen för att returnera MD5-hash av strängen "PostgreSQL MD5":

```
SELECT MD5('PostgreSQL MD5');
```

Resultatet är:

```
               md5
----------------------------------
 f78fdb18bf39b23d42313edfaf7e0a44
(1 row)
```

Förberedelser och konfiguration

Innan libpg kan användas i ett C++-program måste en del förberedelser, konfiguration och anpassning av projektet göras. De förberedelser som görs nedan gäller för CLion med CMake.

Installation av libpq i MacOS

Använd pakethanteraren Homebrew (https://brew.sh/index_sv) för att installera libpq. Exekvera sedan följande kommando i ett terminalfönster:

```
brew install libpq
```

De filer som sedan behövs finner du i:

```
/usr/local/opt/libpq/lib
/usr/local/opt/libpq/include
```

Installation av libpq i Linux

Har du Homebrew installerat sedan tidigare så använd det enligt ovan. Annars kan du installera libpq med följande kommando:

```
sudo apt-get update
sudo apt-get remove libpq5
sudo apt-get install libpq-dev
```

Jag har inte lyckats klura ut var filerna lagras då jag inte har något Linux-system. Du som har Linux får gärna dela med dig via forumet eller e-post var filerna ligger.

Installation av libpq i Windows

Tyvärr finns inget enkelt sätt (vad jag känner till) för att installera libpq i Windows. Du behöver installera PostgreSQL-servern lokalt på din dator för att alla filer ska installeras korrekt. Ladda hem installationsfilen för senaste versionen av PostgreSQL från: https://www.postgresql.org/download/windows/ (välj länken Download tha installer from certified by EnterpriseDB…). I installationsguiden behöver du bara välja att installera PostgreSQL Server och Command Line Tools (Stack Builder behöver vi inte och pgAdmin 4 har du redan installerat). Välj ett master-lösenord för PostgreSQL.

Rotkatalogen för PostgreSQL-installationen på Windows är:

```
C:\Program Files\PostgreSQL\_versionsnr_
```

De filer som behövs för libpq finns i:

```
C:\Program Files\PostgreSQL\_versionsnr_\lib
C:\Program Files\PostgreSQL\_versionsnr_\include
```

För att slippa ha en PostgreSQL-server körandes i bakgrunden hela tiden kan du göra en kopia på de två katalogerna ovan (lib och include). Sen avinstallerar du PostgreSQL (kör uninstall-postgresql.exe som du finner i rotkatalogen och välj Individual components, markera PostgreSQL Server men behåll Command Line Tools). Därefter kopierar du tillbaka lib och include till samma ställe som sökvägarna ovan.

CLion

Vi ska nu prova att allt fungerar. Skapa ett nytt projekt i CLion och i CMakeLists.txt lägger du till följande sist i filen för att inkludera det som behövs för libpq:

```
# This command attempts to find the library, REQUIRED argument is optional
find_package(PostgreSQL REQUIRED)

# Add include directories to your target.
# PRIVATE is useful with multi-target projects
# see documentation of target_include_directories for more info
target_include_directories(PostgreSQLTest PRIVATE ${PostgreSQL_INCLUDE_DIRS})

# Add libraries to link your target against.
# Again, PRIVATE is important for multi-target projects
target_link_libraries(PostgreSQLTest PRIVATE ${PostgreSQL_LIBRARIES})
```

Klicka på reload changes (om du inte har aktiverat auto reload) och förhoppningsvis är allt ok. Om du får ett CMake Error liknande missing: PostgreSQL_INCLUDE_DIR kan du prova att lägga till följande i CMakeLists.txt (innan find_package):

```
# You may need to manually set...
# PostgreSQL_INCLUDE_DIR  - the path to where the PostgreSQL include files are.
# PostgreSQL_LIBRARY_DIR  - The path to where the PostgreSQL library files are.
# PostgreSQL_TYPE_INCLUDE_DIR  - The path to where the pg_type.h file is.
# ... if FindPostgreSQL.cmake cannot find the include files, the library files or the pg_type.h file.
set(PostgreSQL_INCLUDE_DIR "C:/Program Files/PostgreSQL/12/include")
set(PostgreSQL_LIBRARY_DIR "C:/Program Files/PostgreSQL/12/lib")
set(PostgreSQL_TYPE_INCLUDE_DIR "C:/Program Files/PostgreSQL/12/include/server/catalog")
```

Ändra sökvägarna så att de passar för din installation av libpq.

I main.cpp lägger du till följande include:

```
#include <libpq-fe.h>
```

Testa att köra ditt program och förhoppningsvis är allt ok. Uppstår problem kan du fråga efter hjälp i kursens forum.

Nu är du redo att börja med inlämningsuppgiften.