

Praktikum 2

Lernziel

- Erlernen des Konzepts des Datenbankzugriffs über ein natives Call Level Interface.
- Kennenlernen des Transaktionsbegriffs.

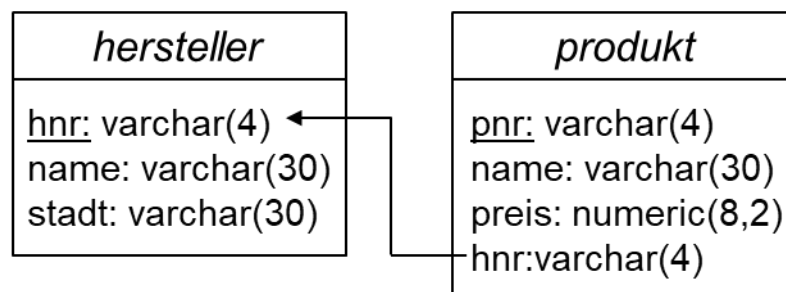
Vorbereitung

Informieren Sie sich über folgende Punkte:

- Wie können in einem C++-Programm die Funktionsdefinitionen auf mehrere Sourcedateien verteilt werden? Welche Schritte werden bis zur Erstellung des Executables durchlaufen (Wiederholung PE1)?
- Wie kann die Projektverwaltung mit make und einem Makefile automatisiert werden? (siehe [1])
- Schauen Sie Sich das bereitgestellte Makefile an. Welche C++-Sourcedateien müssen Sie selber erstellen?
- Schauen Sie Sich die bereitgestellte Datei db.h an. Welche Funktionen müssen Sie in db.cpp implementieren?
- Wie kann man in einem C++-Programm die Kommandozeilenargumente auslesen? (Hinweis: argc, argv)
- Wie kann man einen std::string an eine Funktion wie z.B. PQconnectdb() übergeben, die als Parameter einen const char* erwartet?
- Wie kann man eine Datei in der Programmiersprache C++ zeilenweise einlesen und jede Zeile in die durch ein Trennzeichen getrennten Felder zerlegen? (Hinweis: fgets() und strtok() bzw. strsep())
- Informieren Sie sich über die Verwendung der libpq bei PostgreSQL (siehe [2]).

Aufgabe

Es soll ein kommandozeilen-orientiertes C++-Programm **hpdb** geschrieben werden, mit dessen Hilfe die folgende Hersteller-Produkt-Datenbank (s.u.) verwaltet und manipuliert werden kann. Der Datenbankzugriff innerhalb von hpdb soll über die Bibliothek libpq implementiert werden.



Praktikum 2

Kommandozeilenoptionen von hpdb

Das Programm soll folgendermaßen aus der Linux-Shell aufgerufen werden können:

Usage:

hpdb <connect-options> commandfile

Connect options:

- u *user*: User name
- c *credential*: Password
- h *host*: Hostname of the database server
- p *port*: Port of the database server
- d *database*: Database name

commandfile ist der Pfad zu einer Textdatei, die pro Zeile ein Kommando enthält.

Folgende Kommandos sind möglich:

- n** Erzeugt eine leere Datenbank, indem die Tabellen *produkt* und *hersteller* (falls erforderlich) gelöscht und neu kreiert werden.
- ih** *hnr name stadt*
Fügt ein neues Tupel mit den angegebenen Werten in die Tabelle **hersteller** ein. Falls bereits ein Tupel mit der gegebenen *hnr* existiert, sollen für dieses Tupel die Attribute *name* und *stadt* entsprechend geändert werden.
- ip** *pnr name preis hnr*
Fügt ein neues Tupel mit den angegebenen Werten in die Tabelle *produkt* ein. Falls bereits ein Tupel mit der angegebenen *pnr* existiert, sollen für dieses Tupel die Attribute *name*, *preis* und *hnr* entsprechend geändert werden.
- i** *hnr hname stadt pnr pname preis*
Teilt die angegebenen Werte auf die Tabellen *hersteller* und *produkt* auf, indem entsprechende Tupel eingefügt bzw. aktualisiert werden.
- dh** *hnr*
Löscht das Tupel mit dem Schlüssel *hnr* aus der *hersteller*-Tabelle. Ggf. vorhandene abhängige *produkt*-Tupel sind ebenfalls zu löschen.
- dp** *pnr*
Löscht das Tupel mit dem Schlüssel *pnr* aus der *produkt*-Tabelle.
- ch** Gibt die Anzahl der Tupel in der Tabelle *hersteller* aus.
- cp** Gibt die Anzahl der Tupel in der Tabelle *produkt* aus.

Ein Beispielaufruf der Kommandodatei *test* für den Nutzer *dbs201* mit dem Passwort *12345* auf dem DBS-Praktikumsserver sähe also so aus:

```
hpdb -u dbs201 -c 12345 -h 194.94.121.224 -p 15434 -d dbs201 test
```

Praktikum 2

Funktionalität von hpdb

Das Programm **hpdb** soll sich wie folgt verhalten:

- hpdb verbindet sich über die in `<connect options>` angegebenen Parameter mit dem Datenbankserver.
- hpdb liest die Datei `commandfile` zeilenweise ein und arbeitet die Kommandos nacheinander ab.
- Der komplette Aufruf von hpdb soll in *einer* Transaktion stattfindet, d.h. bei erfolgreicher Abarbeitung aller Kommandos wird nach dem letzten Kommando ein `commit` gemacht. Wenn zwischendurch ein Fehler auftritt, wird die Abarbeitung der Kommandos abgebrochen und alle bis dahin gemachte Änderungen in der Datenbank mittels `rollback` zurückgesetzt.
- Zum Schluss meldet sich hpdb vom Datenbankserver ab.
- Da Datenbank-Operationen aus verschiedenen Gründen fehlschlagen können, gehört zu Ihrem Programm auch eine geeignete Fehlerbehandlung!

Implementierung

Durch das bereitgestellte Makefile ist festgelegt, dass Sie zwei C++ Sourcedateien schreiben müssen: `main.cpp` und `db.cpp`. Die Datei `main.cpp` implementiert die Ablauflogik, d.h. sie parst die Kommandozeilenparameter, liest die Kommandodatei ein und arbeitet sie Zeile für Zeile ab.

Wichtig: In `main.cpp` wird keine einzige `libpq`-Funktion aufgerufen, sondern der Datenbankzugriff erfolgt ausschließlich über die von Ihnen in `db.cpp` zu implementierenden Funktionen. Diese Funktionen sind durch den bereitgestellten Header `db.h` vorgegeben:

`db_login`: *Datenbanklogin*

`db_logout`: *Datenbanklogout*

`db_begin`, `db_commit`, `db_rollback`: *Transaktionsbefehle*

`db_create_table_x`: *Anlegen der Tabellen Hersteller bzw. Produkt*

`db_drop_table`: *Entfernen der Tabellen Hersteller bzw. Produkt*

`db_check_pnr`: *Prüfung ob ein bestimmtes Produkt schon vorhanden ist*

`db_check_hnr`: *Prüfung ob ein bestimmter Hersteller schon vorhanden ist*

`db_insert_hersteller` – *Einfügen eines Datensatz in Tabelle Hersteller*

`db_insert_produk`: *Einfügen eines Datensatz in Tabelle Produkt*

`db_update_hersteller` – *Ändern eines Datensatz in Tabelle Hersteller*

`db_update_produk`: *Ändern eines Datensatz in Tabelle Produkt*

`db_delete_hersteller` – *Löschen eines Datensatzes in Tabelle Hersteller;
abhängige Produkte sollen mitgelöscht werden*

`db_delete_produk` – *Löschen eines Datensatzes in Tabelle Produkt*

Praktikum 2

db_count – Anzahl der Tupel in Hersteller bzw. Produkt zurückgeben;

Die o.g. Funktionen sollen in der Datei `db.cpp` unter Verwendung der CLI-Bibliothek `libpq` implementiert werden. Die dazugehörige Header-Datei `db.h`, in der die Prototypen der `db_x`-Funktionen deklariert sind, steht ebenfalls zum Download bereit.

Test

In der mitgelieferten Archivdatei `hpdbs-tests.tar` finden Sie sieben Kommandodateien `commandfile1`, ..., `commandfile7`, die Sie zum Testen Ihres Programms nutzen können. In der Datei `tests.txt` stehen die erwarteten Datenbankzustände bzw. Ausgaben, wenn Sie die `commandfiles` nacheinander zur Ausführung bringen.

Literatur

[1] Learn Makefiles. Online verfügbar unter <https://makefiletutorial.com/>

[2] PostgreSQL Programmer's Guide: Client Interfaces - libpq. Im PG-Paket enthalten. Online verfügbar unter <https://www.postgresql.org/docs/13/libpq.html>