Adatbázis használata Qt-vel

Adatbázisok használatának Qt-s támogatása

- A Qt-ben a *QtSql* module támogatja az SQL elérésű adatbázisok platform- és adatbázis független használatát.
- Az adatbázis kapcsolat kiépítése során ún. meghajtók (driver-ek) biztosítják a különböző adatbázisok API-jaival való kommunikációt.
- Az SQL szintaxist jól ismerők számára lehetőség van arra, hogy Qt-ben közvetlenül SQL utasítások segítségével végezzék az adatbázis-kezelést.
- A magasabb szintű adatbázis-kezelést kedvelők az adatbázis adatainak egy részét ún. SQL táblákba tudják betölteni, majd ezekben, mint adatmodellekben, az adatokat bejárhatják és szerkeszthetik, végül visszaírhatják azokat az adatbázisba. Ezek az adatmodellek megjelenítő widget-ekkel párosíthatóak.

Adatbázis meghajtók

- A QtSql modul több beépített meghajtót is tartalmaz a különböző adatbázis-motorok kezelésére, valamint felületet biztosít további meghajtók készítésére:
  - QMYSQL ~ MySQL
  - QOCI ~ Oracel (Oracel call interface driver)
  - QODBC ~ ODBC (open database connectivity)
    pl. Microsoft SQL Server-hez
  - QSQLLITE ~ SQLite
- Nincs minden meghajtó előre telepítve, bizonyos esetekben további csomagként (pl. **`libqt5sql5-mysql`**) kell hozzáadnunk őket.

A QtSql modul használatba vétele

- A modul osztályainak használatához a megfelelő osztály könyvtárát kell „beinklúdolni” az aktuális fájlban, illetve lehetőség van a teljes modul betöltésére is:

  ```
  #include <QtSql>
  ```

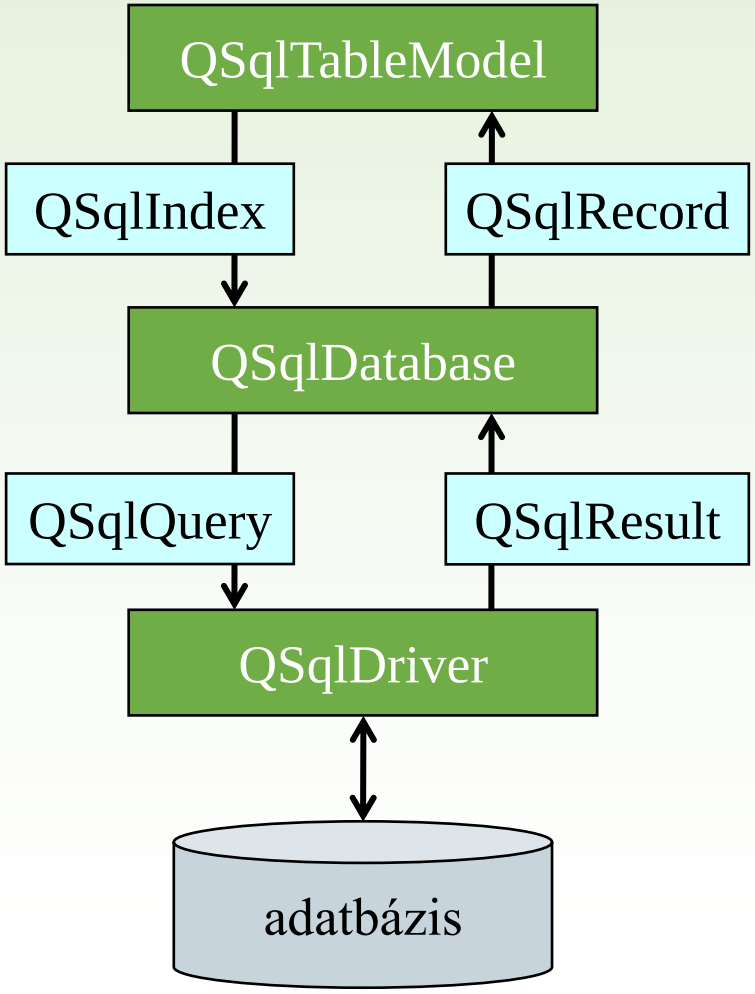
- A modul csak akkor érhető el egy Qt alkalmazásban, ha használatát a projektfájlban jelezzük a `QT += sql` utasítással
  - a betöltendő modulok egy sorban is lehetnek, pl.
    `QT += core gui widgets sql`
  - amennyiben konzolon szerkesztünk, a projektfájl létrehozásakor is hozzáadhatjuk a modult:

    ```
    qmake -project "QT += sql"
    ```

A Qt adatbázis-kezelő modul

A *QtSql* modul osztályai:

- A modell osztályok biztosítják a logikai (memóriabeli) adatbázist: **`QSqlQueryModel`**, **`QSqlTableModel`**, **`QSqlRelationalModel`**.
- Az (API) alkalmazásprogramozói osztályok biztosítják az SQL elemek kezelését: **`QSqlDataBase`**, **`QSqlQuery`**, **`QSqlError`**, **`QSqlResult`**, **`QSqlField`**, **`QSqlIndex`**, **`QSqlRecord`**.
- A meghajtó osztályok biztosítják az adatbázis elérését és a kommunikációt: pl. **`QSqlDriver`**



Adatbáziskapcsolat létesítése

- Adatbázismotor betöltése, és a kapcsolatot létesítése a **`QSqlDatabase`** osztály segítségével történik, pl.:

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
db.setHostName("localhost");        // szerver
db.setDatabaseName("myDatabase"); // adatbázis
db.setUserName("root");             // felhasználónév
db.setPassword("root");             // jelszó
```

meghajtó

- Lehetőségünk van egyszerre több kapcsolatot is kezelni a programban:
  - A kapcsolatokat egyedi nevekkel láthatjuk el: **`addDatabase(`*`<meghajtó>`*`, `*`<név>`*`)`**.
  - A kapcsolatok nevei listázhatóak (**`connectionNames()`**), név alapján egy kapcsolat lekérhető (**`database(`*`<név>`*`)`**), vagy törölhető (**`removeDatabase(`*`<név>`*`)`**).
- Egy kapcsolatot az **`open()`** nyit meg, a **`close()`** zár be.
  - ha nem sikerül a megnyitás, hamissal tér vissza
  - a program bezárása nem zárja be a nyitott kapcsolatokat

SQL utasítások végrehajtása

- SQL utasítást közvetlenül az **`QSqlQuery`** segítségével futtathatunk, pl.:

```
QSqlQuery query;
if (query.exec("select id, data from myTable"))
        // parancs futtatása
        // … a query által visszaadott válasz
        // soronkénti értelmezése
else // sikertelen futtatás
    cout << query.lastError().text();
```

  - Ennek konstruktorában megadható paraméterként az adatbázis kapcsolat, de ha nem adjuk meg, az alapértelmezettet használja.
  - Az **`exec()`** művelettel tetszőleges utasítást végrehajthatunk, és igazzal tér vissza, amennyiben sikerült azt végrehajtania.
  - A **`lastError()`** segítségével lekérdezhetjük a végrehajtási hiba okát.

Lekérdezés olvasása

- A lekérdezés eredménye mindig egy táblázat, amelynek egyszerre csak egy sorát láthatjuk.
  - A **`size()`** megadja a lekérdezett sorok számát.
  - Lépegetni a **`first()`**, **`next()`**, **`previous()`**, **`last()`** utasításokkal lehet (igazat adnak, ha tudnak lépni). Kezdetben az eredmény első sora előtt állunk, tehát kezdéshez rögtön léptetni kell.
  - Amennyiben csak előre akarunk lépkedni, a **`setForwardOnly()`** metódus optimalizálja a lekérdezést.
  - Adott sorra ugrani a **`seek(<sorszám>)`** metódussal.
  - Értéket lekérdezni a kijelölt sorból a **`value(<oszlopszám>)`** metódussal tudunk. Az érték **`QVariant`** típusú, így az tovább konvertálható alkalmas formára.

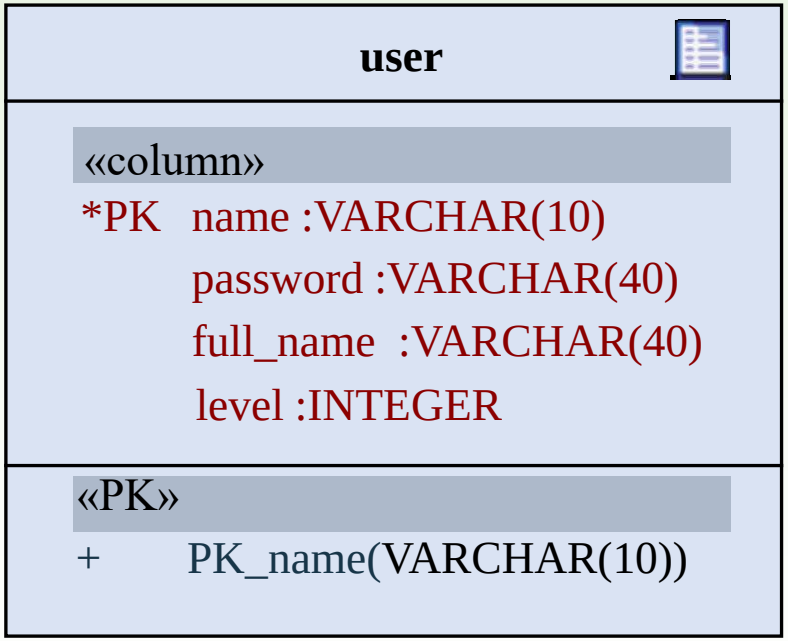
```
while (query.next()){
    cout << query.value(0).toString(); // első oszlop szöveg
    cout << query.value(1).toInt();    // második oszlop egész szám
}
```

1.Feladat

Készítsünk egyszerű konzol alkalmazást, amely lehetővé teszi egy adatbázis (**`sample`**) felhasználók táblájának (**`user`**) listázását, egy felhasználójának törlését, valamint új felhasználó felvételét.

- A programot menün keresztül vezéreljük (ehhez hozzunk létre egy menü osztályt).
- A program csak megfelelő felhasználónév/jelszó megadásával engedi elvégezni a kért tevékenységeket.
- A program SQL lekérdezés segítségével azonosítja a felhasználót, SQL paranccsal olvassa be a táblatartalmat (azonosító, név, jelszó, szint), hogy kilistázza, és SQL paranccsal ad lehetőséget az azonosító alapján történő törlésre és beszúrásra.

1.Feladat: tervezés



1.Feladat: tervezés

```
DROP DATABASE IF EXISTS sample;
CREATE DATABASE sample;
USE sample;

CREATE TABLE user(
     name VARCHAR(10) PRIMARY KEY,
     password VARCHAR(40) NOT NULL,
     full_name VARCHAR(40) NOT NULL,
     level INTEGER NOT NULL
);

INSERT INTO user(name, password, full_name, level)
          VALUES('root', 'root', 'admin', 0);
INSERT INTO user(name, password, full_name, level)
          VALUSE('gt', 'secret', 'Gregorics', 1);
```

name	password	full_name	level
root	root	admin	0
gt	secret	Gregorics	1

1.Feladat: adatbázis kapcsolat

```
int main(int argc, char *argv[]){
    QCoreApplication a(argc, argv);
    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    db.setHostName("localhost");
    db.setDatabaseName("sample");
    db.setUserName("root");
    db.setPassword("root");
    if (db.open()) {
        Menu m;
        m.run();
        db.close();
    }
    else { … }
    return 0;
}
```

adatbáziskapcsolat kiépítése

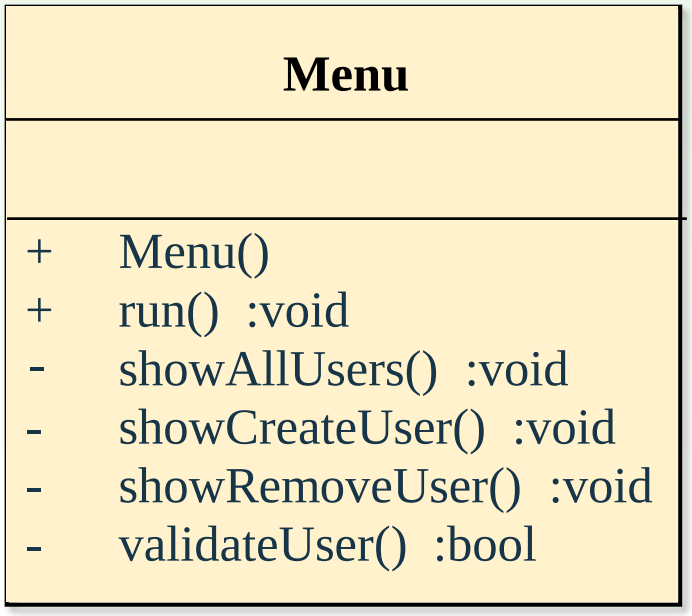
sikeres megnyitás esetén

kapcsolat bezárása

sikertelen kapcsolódás

1.Feladat: Menu osztály

- Hozzunk létre egy menü osztályt egy menü kezelésére. A menüt futtató **`run()`** metóduson kívül legyen benne az azonosítás végző (**`validateUser()`**) metódus és a három menüpont metódusa.



1.Feladat: listázás

```
void Menu::showAllUsers(){
    …  // fejléc írása a konzolra
    QSqlQuery selectQuery;

    selectQuery.exec("select name, password, full_name, level from user");

    while (selectQuery.next()) {
        cout
        << setw(20) << selectQuery.value(0).toString().toStdString()
        << setw(20) << selectQuery.value(1).toString().toStdString()
        << setw(20) << selectQuery.value(2).toString().toStdString()
        << setw(20) << selectQuery.value(3).toInt() << endl;
    }
}
```

SQL parancs és végrehajtása

eredmény lekérdezése

1.Feladat: beszúrás

```
void Menu::showCreateUser(){
     string name, fullName, password;
     int level;
     …
     QSqlQuery insertQuery;
     insertQuery.exec("insert into user values('" +
          QString::fromStdString(name)     + "', '" +
          QString::fromStdString(fullName) + "', '" +
          QString::fromStdString(password) + "', " +
          QString::fromStdString(level)    + ")");
     if (!insertQuery.exec()) // ha sikertelen a végrehajtás
          cout << "Hiba történt: "
               << insertQuery.lastError().text().toStdString() << endl;
}
```

beszúrandó adatok megszerzése

SQL parancs és végrehajtása

```
INSERT INTO user(name, password, full_name, level)
          VALUES('…', '…', '…', 0)
```

1.Feladat: törlés

```
void Menu::showRemoveUser() {
    string name;
    cout << "Azonosító: ";
    getline(cin, name);

    QSqlQuery removeQuery;
    removeQuery.exec("delete from user where name = '" +
        QString::fromStdString(name) + "'");
}
```

SQL parancs és végrehajtása

```
DELETE FROM user WHERE name = '…'
```

1.Feladat: azonosítás

```
bool Menu::validateUser(){
    string name, password;
    cout << "Felhasználónév: "; getline(cin, name);
    cout << "Jelszó: "; getline(cin, password);

    QSqlQuery selectQuery;
    selectQuery.exec(
    "select name, password from user where name = '"
    + QString::fromStdString(name) + "' and password = '"
    + QString::fromStdString(password) + "'");
    return selectQuery.next();
}
```

adatok beolvasása

SQL parancs és végrehajtása

ha van eredmény akkor sikerült a lekérdezés

SQL injekció

- Az adatbázisban tárolt adatokat meg kell óvni az illetéktelen felhasználók elől, biztonságossá kell tenni az adatokhoz való hozzáférést.
  - A programok az adatbázis-szerveren SQL lekérdezéseket futtatnak, amelyeket szövegként állítanak össze.
  - Az összeállítás során törekedni kell arra, hogy ne lehessen úgy manipulálni az utasítást, hogy az rejtenivaló adatokat tegyen láthatóvá vagy kárt tegyen a tárolt adatokban.
- Az SQL parancsok manipulációját nevezzük *SQL injekció*nak (*SQL injection*)
  - leggyakrabban webes környezetben fordul elő

Példa SQL injekcióra

```
// felhasználónév/jelszó
    string name, password;
    cout << "Felhasználónév: "; getline(cin, name);
    cout << "Jelszó: "; getline(cin, password);
    QSqlQuery query;
    query.exec("select name, password from user where name = '"
        + QString::fromStdString(name) + "' and password = '"
        + QString::fromStdString(password) + "'");
```

name = "gt" és **password = "secret"** esetén:

```
query.exec("select name, password from user where
    name = 'gt' and password = 'secret'");
```

name = "" és **password = "' or '1' = '1"** esetén:

```
query.exec("select name, password from user where
    name = '' and password = '' or '1' = '1'");
```

A where feltétel azonosan igaz, ezért a program a tábla összes nevét kilistázza.

SQL injekció kivédése

1. A felhasználótól kapott értékek ellenőrzése, módosítása.

```
name = name.remove("'");
```

2. A felhasználó által megadható adatok korlátozása.

```
QLineEdit linEdit;
lineEdit.setInputMask("aaaaaaaaaaaaaaa");
    // csak alfabetikus karaktereket fogad el
lineEdit.setValidator(QRegExpValidator("[A-Za-z0-9]{1,8}"));
    // 1-8 db alfanumerikus karaktert fogad el
```

3. Paraméteres utasítások használata (pl. ORACLE stílusú helyőrző nevekkel építjük fel az SQL parancsot).

```
QSqlQuery query;
query.prepare("select name from user where " +
                  "name = :uname and password = :psw");
query.bindvalue(":uname" , "akarki"); // paraméter beírása
query.bindvalue(":psw"   , "admin");
query.exec();
```

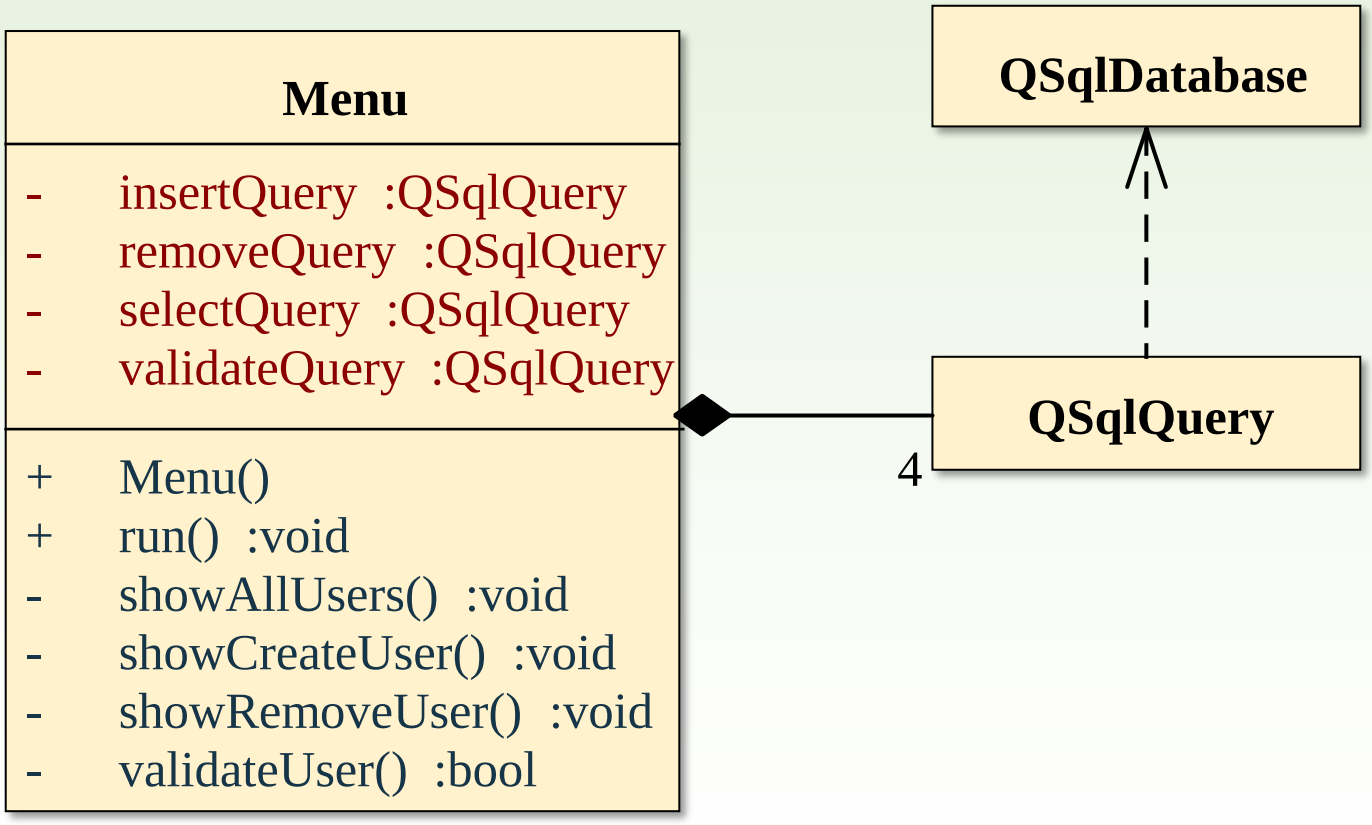
4. Az injekciót eleve kizáró megoldások (pl. adatmodell) használata.

2.Feladat

Készítsünk egyszerű konzol alkalmazást, amely alkalmas egy adatbázis (**`sample`**) felhasználók (**`user`**) táblájának listázására, felhasználó törlésére, valamint új felhasználó beszúrására. A program

- vezérlése menün keresztül történik,
- csak megfelelő felhasználónév/jelszó megadásával engedi elvégezni a tevékenységeket,
- SQL lekérdezés segítségével azonosítja a felhasználót, olvassa be a táblatartalmat (azonosító, név, jelszó, szint), és ad lehetőséget az azonosító alapján történő törlésre és beszúrásra,
- paraméteres utasításokat fog használni kikerülve az SQL injekció lehetőségét.

2.Feladat: tervezés



2.Feladat: megvalósítás

```
bool Menu::validateUser(){
    string name, password;
    cout << "Felhasználónév: ";
    getline(cin, name);
    cout << "Jelszó: ";
    getline(cin, password);
    validateQuery.bindValue(":name", QString::fromStdString(name));
    validateQuery.bindValue(":password",
                            QString::fromStdString(password));
    validateQuery.exec();
    return validateQuery.next();
}
```

adatok beolvasása

paraméterek behelyettesítése

lekérdezés futtatása

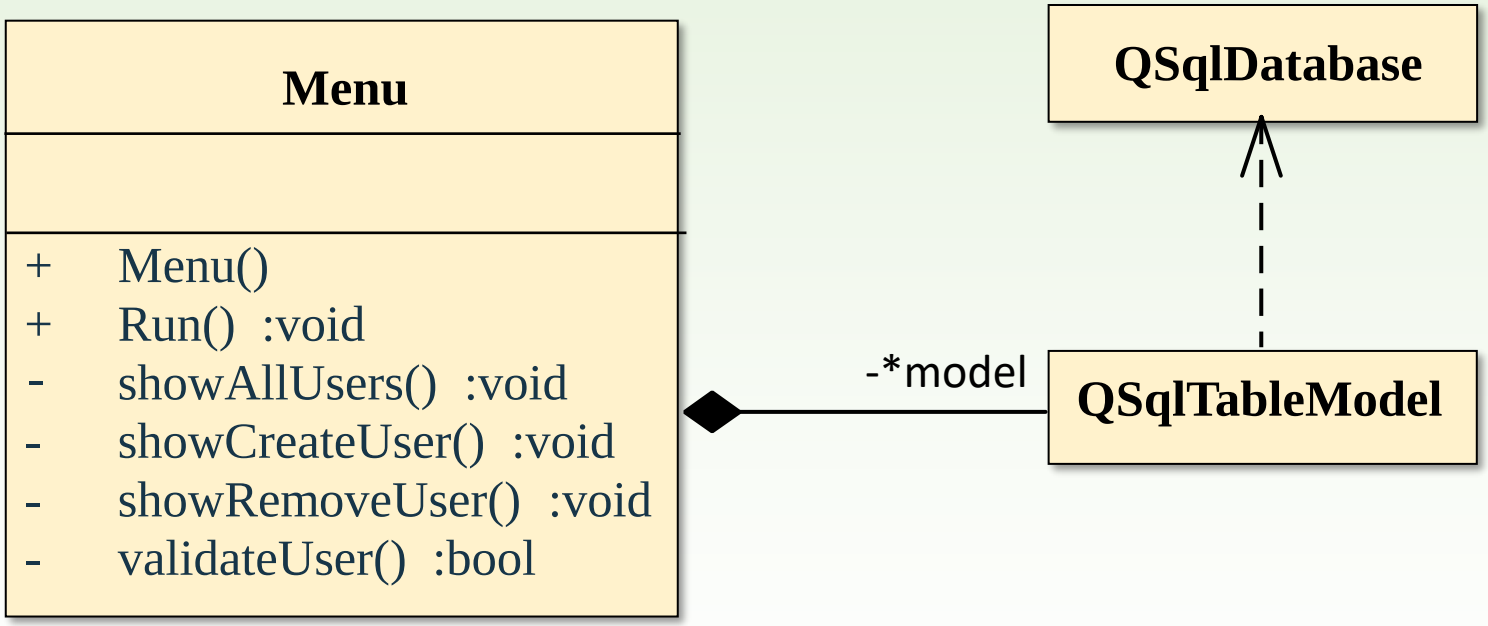
ha van eredmény akkor sikerült a lekérdezés

3.Feladat

Készítsünk egyszerű konzol alkalmazást, amely alkalmas egy adatbázis (**`sample`**) felhasználók (**`user`**) táblájának listázására, felhasználó törlésére, valamint új felhasználó beszúrására.

- Az adatok kezelését egy táblamodellel végezzük (**`QSqlTableModel`**), amely a háttérben futtatja a megfelelő SQL műveleteket.

3.Feladat: tervezés



3.Feladat: megvalósítás

```
bool Menu::Menu(){
    …
    model = new QSqlTableModel();
    model->setTable("user");
}
```

tábla modell beállítása

```
void Menu::ShowAllUsers(){
    model->select();
    for(int i=0; i < model->rowCount(); ++i){
        QSqlRecord record = model->record(i);
        cout << record.value(name).toString()
             << record.value(password).toString() << endl;
    }
}
```

i-edik rekord

adatbázisbeli mezőnév

3.Feladat: megvalósítás

```
void Menu::ShowCreateUser(){
    int row = 0;
    model->insertRows(row, 1);
    model->setData(model->index(row, 0), "gt" );
    model->setData(model->index(row, 1), "secret" );
    model->setData(model->index(row, 2), "Gregorics" );
    model->setData(model->index(row, 3), 0);
    model->SubmitAll();
}
```

1 új sort szúr be a modellbe

adatbázisba mentés

3.Feladat: megvalósítás

```
void Menu::ShowRemoveUser(){
    string name;
    cout << "Felhasználónév: "; getline(cin, name);
    Qstring filter = "name = " + QString::fromStdString(name);
    model->setFilter(filter);
    model->select();
    if(model->rowCount() > 0){
        model->removeRows(0, model->rowCount());
        model->SubmitAll();
    }
}
```

szűrő beállítása a modell feltöltéséhez

adatbázisba mentés

3.Feladat: megvalósítás

```
bool Menu::validateUser(){
    string name, password;
    cout << "Felhasználónév: "; getline(cin, name);
    cout << "Jelszó: "; getline(cin, password);
    Qstring filter = "name = " + QString::fromStdString(name);
    model->setFilter(filter);
    model->select();
    return model->rowCount()>0;
}
```

szűrő beállítása a modell feltöltéséhez