

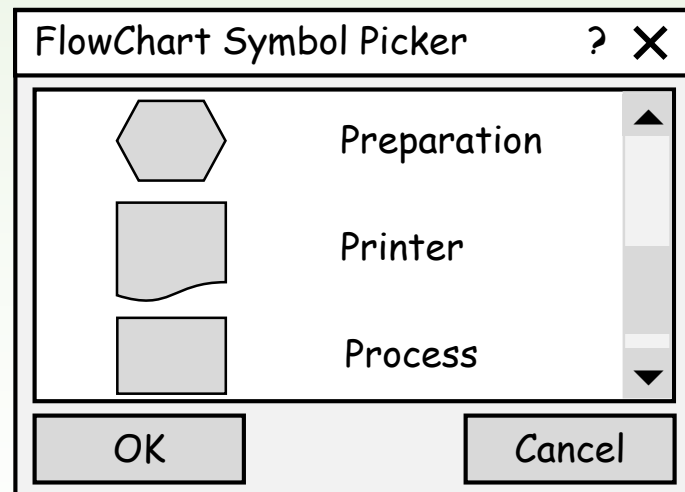
Adatok megjelenítése

Adattárolás a grafikus vezérlőben

- ❑ Számos grafikus vezérlő objektum adatokat is tartalmaz.
 - Ilyen például a `QLineEdit`, `QLCDNumber`, de akár a `QLabel` is,
 - ugyanakkor vannak adathalmazok megjelenítésre és szerkesztésre specializálódott grafikus vezérlő objektumok (*view widget*) is, mint például a `QListWidget`, `QTableWidget`, `QTreeWidget`.
- ❑ A grafikus vezérlőkben tárolt adatokat a vezérlők által kiváltott szignálok slot-jai manipulálhatják, de az alkalmazás bármelyik olyan kódrészlete is, amely hivatkozhat a vezérlő objektumra.

1.Feladat

Készítsünk olyan dialógus alkalmazást, amely folyamatábra rajzolásához szükséges képelemek listáz ki, és lehetővé teszi, hogy ezek közül egyet kiválasszunk, amelynek a sorszámát adja majd vissza az alkalmazás.



1.Feladat: tervezés

- Az alkalmazás számára származtatunk egy dialógus osztályt osztályt (**FlowChartSymbolPicker**), felhelyezünk rá lista widget-et (**QListWidget**) és két nyomógombot (**QPushButton**).

<i>QDialog</i>	
FlowChartSymbolPicker	
-	<code>_listWidget :QListWidget*</code>
-	<code>_okButton :QPushButton*</code>
-	<code>_cancelButton :QPushButton*</code>
-	<code>_id :int</code>
+	<code>FlowChartSymbolPicker(QWidget*)</code>
+	<code>selectedId(): int { return _id; }</code>
+	<code>done()</code>

1.Feladat: megvalósítás

```
#include <QApplication>
#include "flowchartsymbolpicker.h"
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QMap<int, QString> symbolMap;
    symbolMap.insert(132, QObject::tr("Data"));
    symbolMap.insert(135, QObject::tr("Decision"));
    symbolMap.insert(137, QObject::tr("Document"));
    symbolMap.insert(138, QObject::tr("Manual Input"));
    symbolMap.insert(139, QObject::tr("Manual Operation"));
    symbolMap.insert(141, QObject::tr("On Page Reference"));
    symbolMap.insert(142, QObject::tr("Predefined Process"));
    symbolMap.insert(145, QObject::tr("Preparation"));
    symbolMap.insert(150, QObject::tr("Printer"));
    symbolMap.insert(152, QObject::tr("Process"));

    FlowChartSymbolPicker picker(symbolMap);
    picker.show();
    return app.exec();
}
```

1.Feladat: megvalósítás

```
FlowChartSymbolPicker::FlowChartSymbolPicker(
    const QMap<int, QString> &symbolMap, QWidget *parent) : QDialog(parent)
{
    _id = -1;
    _listWidget = new QListWidget;
    _listWidget->setIconSize(QSize(60, 60));

    QMapIterator<int, QString> i(symbolMap);
    while (i.hasNext()) {
        i.next();
        QListWidgetItem *item = new QListWidgetItem(i.value(), _listWidget);
        item->setIcon(iconForSymbol(i.value()));
        item->setData(Qt::UserRole, i.key());
    }
    _okButton = new QPushButton(tr("OK"));
    _okButton->setDefault(true);
    _cancelButton = new QPushButton(tr("Cancel"));

    connect(_okButton, SIGNAL(clicked()), this, SLOT(accept()));
    connect(_cancelButton, SIGNAL(clicked()), this, SLOT(reject()));


    // elrendezők definiálása
    ...
}
```

itt kerülnek be az adatok
a _listWidget-be

1.Feladat: megvalósítás

```
void FlowChartSymbolPicker::done(int result)
{
    _id = -1;
    if (result == QDialog::Accepted) {
        QListWidgetItem *item = _listWidget->currentItem();
        if (item) _id = item->data(Qt::UserRole).toInt();
    }
    QDialog::done(result);
}

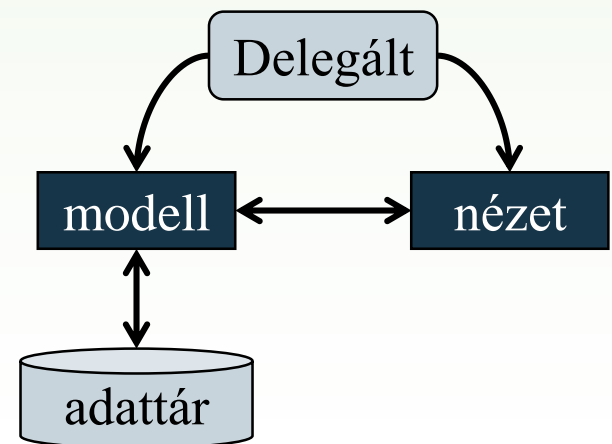
QIcon FlowChartSymbolPicker::iconForSymbol(const QString &symbolName)
{
    QString fileName = ":/images/" + symbolName.toLowerCase();
    fileName.replace(' ', '-');
    return QIcon(fileName);
}
```



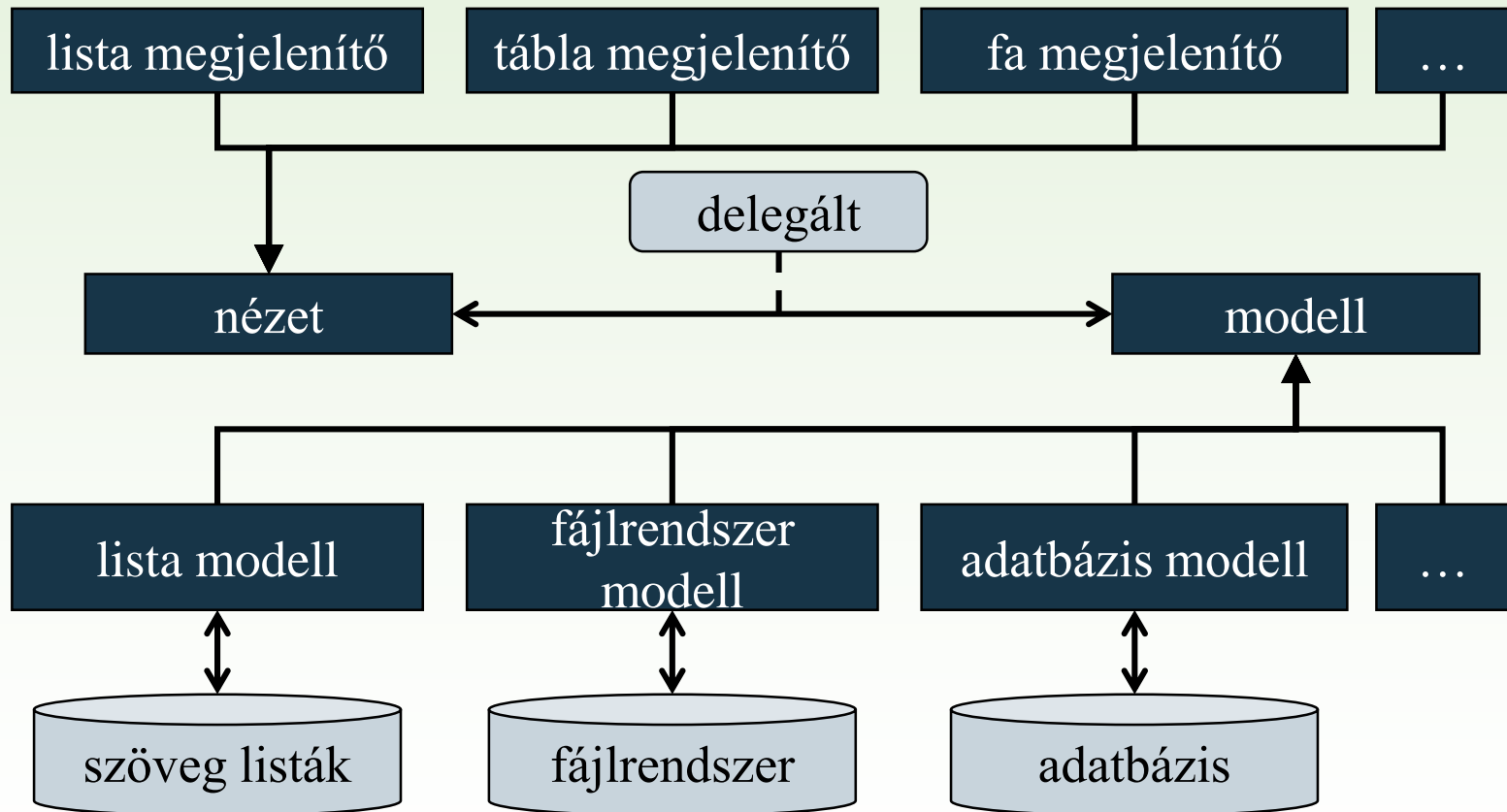
a dialógus megjegyzi a
kiválasztott listaelem sorszámát

Nézet és a modell elválasztása

- ❑ A grafikus vezérlő objektumokban történő adattárolás kényelmetlen
 - ha ugyanazon adatokat más vezérlők is használják, mert ilyenkor gondoskodni kell a szinkronizálásról. Ráadásul ugyanazon adat többszörös nyilvántartása memória pazarlási problémákat is felvet.
 - Ha a grafikus vezérlőben tárolt adat egy nagyobb, háttérben tárolt adathalmaz része.
- ❑ Az adatok csoportos kezelése során célszerű elválasztani az adatok tárolását (modell) azok megjelenítésétől (nézet), azaz az M/V (model/view) architektúrát alkalmazzuk.
- ❑ A modell-nézet szinkronizációt általánosan az ún. MVC (*model-view-controller*) architektúra biztosíthatja. Qt-ben ennek egy olyan változata került megvalósításra, amelyben a modellben tárolt adatok megjelenítésének illetve szerkesztésének módját ún. delegált (*delegate*) osztály írja le.



Csoportos adatkezelés és megjelenítés eszközei

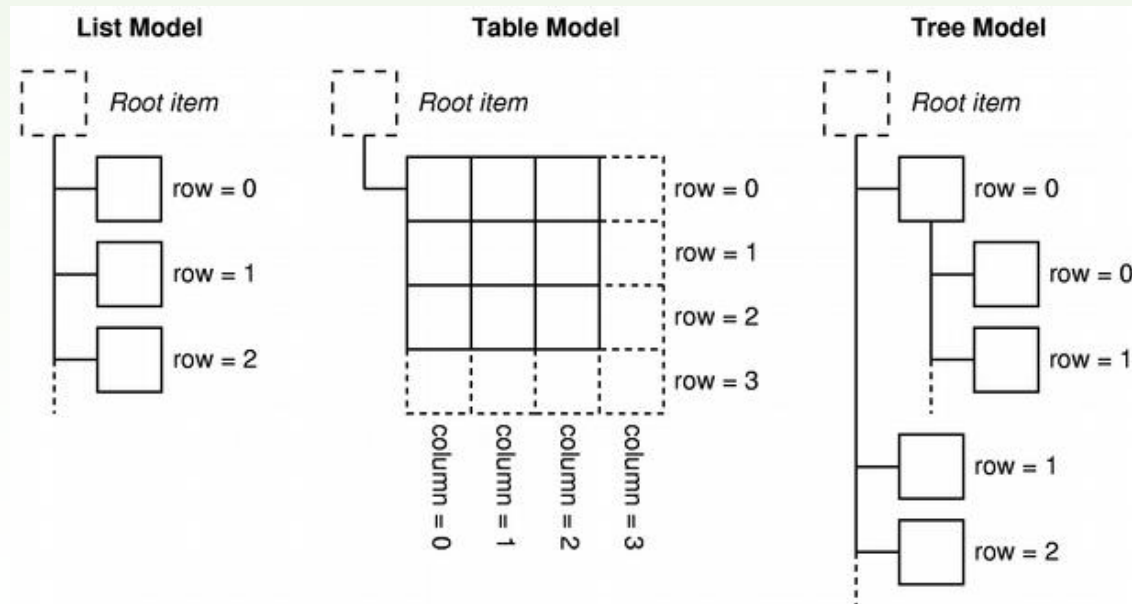


Adatkezelés MV architektúrában

- ❑ Az M/V architektúrában az adatok csoportos megjelenítéshez a például a `QListView`, `QTableView`, `QTreeView` osztályokat használhatjuk. Ezek rendelkeznek egy `model` adattaggal, amely a vezérlőben megjelenő adatot hordozó modell objektumra hivatozik.
- ❑ Az adatok csoportos tárolására különféle modell osztályok állnak rendelkezésünkre
 - `QAbstractItemModel` és az abból származtatott absztrakt `QAbstractListModel` és `QAbstractTableModel`
 - `QStandardItemModel`, `QStringListModel`, `QDirModel`, `QSqlQueryModel`, `QSqlTableModel`
- ❑ A nézet adatmezőiben a modell adatelemei értékeinek megjelenési módját a *delegált* (`QAbstractItemDelegate` leszármazott) osztályok biztosítják. Az előre definiált delegált osztályok közül az alap megjelenítést a `QItemDelegate` szolgáltatja. Az alapértelmezett beállítás sok esetben elegendő.

Modell elemeinek indexelése

- ❑ A *modell indexek* (**QModelIndex**) a (lista, táblázat, fa) modellbeli adatelemek lokalizálására szolgálnak.
 - Az indexhez tartozó adatelemnek mindig van egy sorszáma (**row()**), táblázat esetén oszlopszáma (**column()**) is, fák esetén gyerekei, és szülője (**parent()**).
 - Egy index **data()** metódusa a megfelelő adatelemre hivatkozik.



Modell indexek a nézetben

□ Az indexeket a nézetben is használhatjuk.

- beállítható az adatelemek kiválasztásának módja:
 - `setSelectionBehavior()` – egy klikkelés elemet, sort, vagy oszlopot választ
 - `setSelectionMode()` – egyszeres vagy csoportos kiválasztás engedélyezése
- aktuális elem kijelölése: `setCurrentIndex(<index>)`
- az `edit(<index>)` művelettel szerkeszthetővé tehetünk egy elemet, az `update(<index>)` frissíti az adott tartalmat.

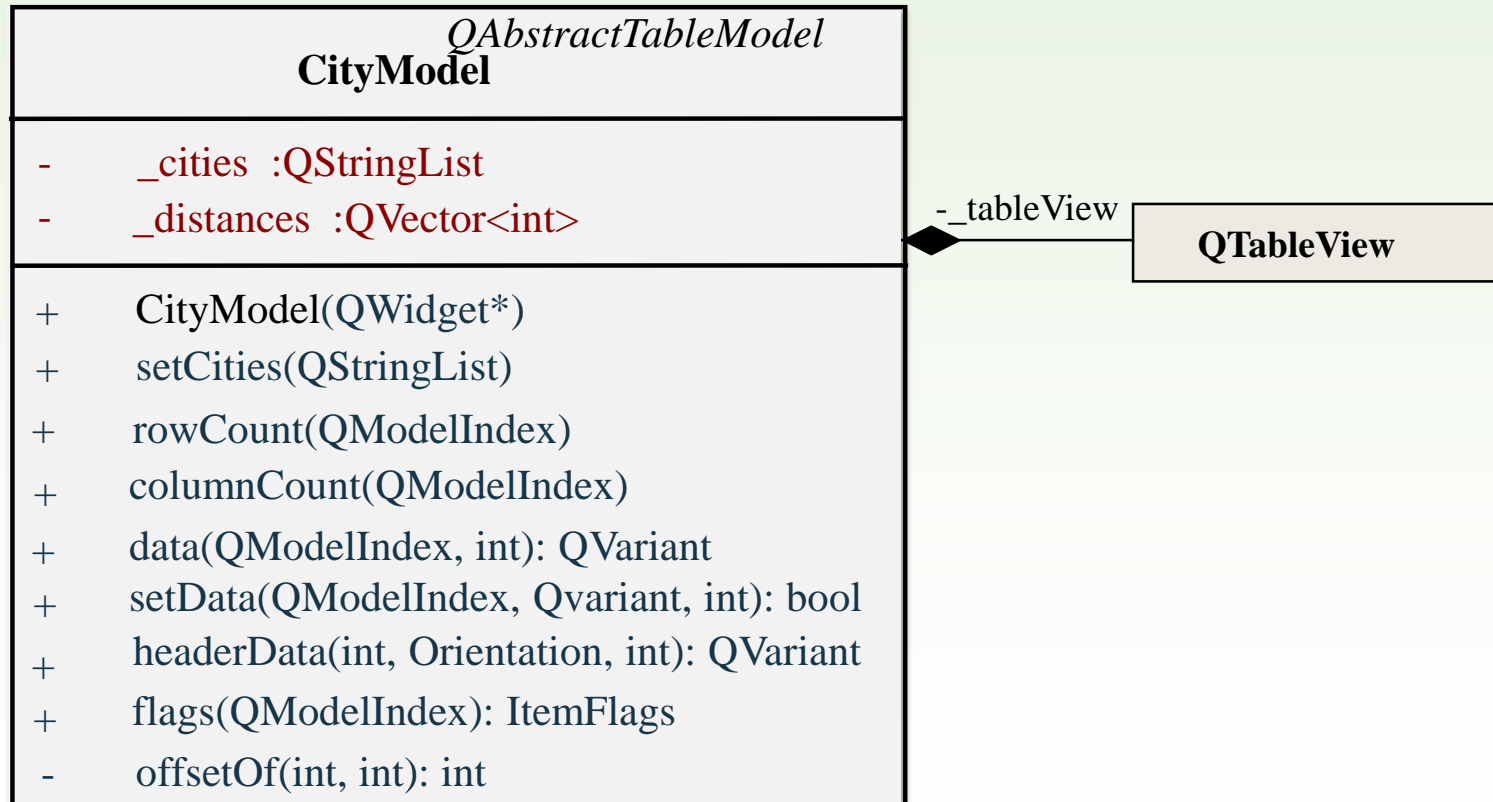
2.Feladat

Készítsünk olyan alkalmazást, amelyikben adott városok távolságait lehet feltüntetni.

Városok						
	Eger	Kaposvár	Győr	Kecskemét		
Eger	0	210	300	0		
Győr	210	0	0	0		
Kaposvár	300	0	0	0		
Kecskemét	0	0	0	0		
Miskolc	0	0	0	0		

2.Feladat: tervezés

- A megoldáshoz egy táblamodell osztályt készítünk, amely egy **QTableView** segítségével jeleníti meg az adatokat.



2.Feladat: megvalósítás

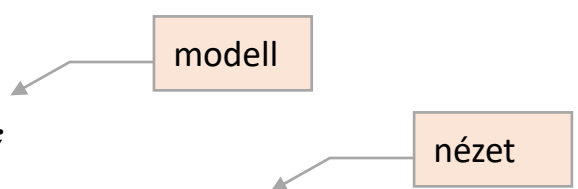
```
#include <QApplication>
#include "citymodel.h"
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    QStringList cities;
    cities << "Békéscsaba" << "Budapest" << "Debrecen" << "Eger"
        << "Győr" << "Kaposvár" << "Kecskemét" << "Miskolc"
        << "Nyíregyháza" << "Pécs" << "Salgótarján" << "Szeged"
        << "Szekszárd" << "Székesfehérvár" << "Szolnok"
        << "Szombathely" << "Tatabánya" << "Veszprém"
        << "Zalaegerszeg";

    CityModel cityModel;
    cityModel.setCities(cities);

    QTableView tableView; tableView.setModel(&cityModel);
    tableView.setAlternatingRowColors(true);
    tableView.setWindowTitle(QObject::tr("Városok")); tableView.show();

    return app.exec();
}
```



```
graph LR
    modell[modell] --> CityModel
    nezet[nézet] --> QTableView
```

2.Feladat: megvalósítás

```
CityModel::CityModel(QObject *parent) : QAbstractTableModel(parent) { }

void CityModel::setCities(const QStringList &cityNames)
{
    _cities = cityNames;
    _distances.resize(_cities.count() * (_cities.count() - 1) / 2);
    _distances.fill(0);
    resetInternalData();
}

int CityModel::rowCount(const QModelIndex & ) const
{
    return _cities.count();
}

int CityModel::columnCount(const QModelIndex & ) const
{
    return _cities.count();
}
```

n város esetén $n(n-1)/2$ távolságot kell tárolni

felüldefiniálандó metódusok

2.Feladat: megvalósítás

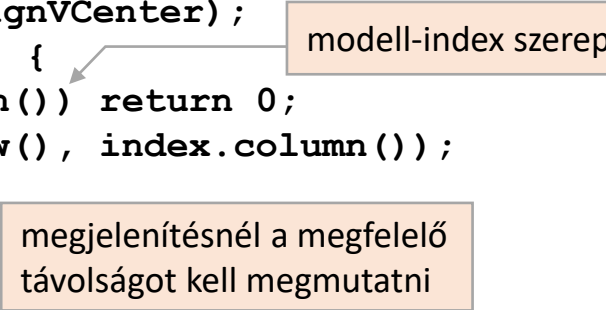
```
QVariant CityModel::headerData(int section, Qt::Orientation, int role)
const
{
    if (role == Qt::DisplayRole) return _cities[section];
    return QVariant();
}
Qt::ItemFlags CityModel::flags(const QModelIndex &index) const
{
    Qt::ItemFlags flags = QAbstractItemModel::flags(index);
    if (index.row() != index.column()) flags |= Qt::ItemIsEditable;
    return flags;
}
int CityModel::offsetOf(int row, int column) const
{
    if (row < column) qSwap(row, column);
    return (row * (row - 1) / 2) + column;
}
```

a fejléc megfelelő pozícióján megjelenő adat

csak a táblázat átlóján kívüli elemek nem szerkeszthetők

2.Feladat: megvalósítás

```
QVariant CityModel::data(const QModelIndex &index, int role) const
{
    if (!index.isValid()) return QVariant();
    if (role == Qt::TextAlignmentRole) {
        return int(Qt::AlignRight | Qt::AlignVCenter);
    } else if (role == Qt::DisplayRole) {
        if (index.row() == index.column()) return 0;
        int offset = offsetOf(index.row(), index.column());
        return _distances[offset];
    }
    return QVariant();
}
```



modell-index szerepe

megjelenítésnél a megfelelő távolságot kell megmutatni

2.Feladat: megvalósítás

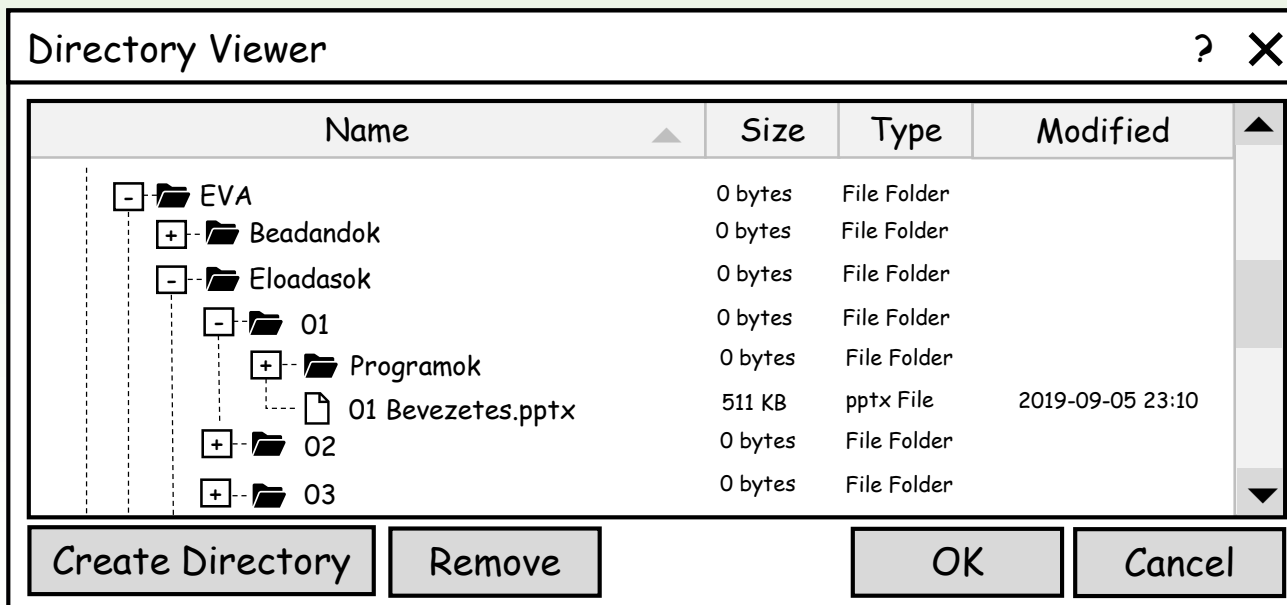
```
bool CityModel::setData(const QModelIndex &index,
                        const QVariant &value, int role) {
    if (index.isValid() &&
        index.row() != index.column() && role == Qt::EditRole) {
        int offset = offsetOf(index.row(), index.column());
        _distances[offset] = value.toInt();
        QModelIndex transposedIndex =
            createIndex(index.column(), index.row());
        emit dataChanged(index, index);
        emit dataChanged(transposedIndex, transposedIndex);
        return true;
    }
    return false;
}
```

modellindex szerepe

szerkesztésnél a megfelelő távolságot
tárolni kell és a tábla szimmetrikus
elemében megjeleníteni

3.Feladat

Készítsünk könyvtárkezelő alkalmazást.



3.Feladat: tervezés

- A megoldáshoz egy dialógus osztályt készítünk (**DirectoryViewer**), felhelyezünk rá fanézetet (**QTreeView**) és egy könyvtármodellt (**QDirModel**), valamint két nyomógombot (**QPushButton**).

DirectoryViewer <i>QDialog</i>	
-	<code>_treeView :QTreeView*</code>
-	<code>_model :QDirModel*</code>
-	<code>_mkdirButton :QPushButton*</code>
-	<code>_removeButton :QPushButton*</code>
+	<code>DirectoryViewer(QWidget*)</code>
slots:	
-	<code>createDirectory()</code>
-	<code>remove()</code>

3.Feladat: megvalósítás

```
DirectoryViewer::DirectoryViewer(QWidget *parent) : QDialog(parent)
{
    _model = new QDirModel;
    _model->setReadOnly(false);
    _model->setSorting(QDir::DirsFirst | QDir::IgnoreCase | QDir::Name);

    _treeView = new QTreeView; treeView->setModel(_model);
    _treeView->header()->setStretchLastSection(true);
    _treeView->header()->setSortIndicator(0, Qt::AscendingOrder);
    _treeView->header()->setSortIndicatorShown(true);
    _treeView->header()->setSectionsClickable(true);

    QModelIndex index = _model->index(QDir::currentPath());
    _treeView->expand(index);
    _treeView->scrollTo(index);
    _treeView->resizeColumnToContents(0);

    ...
}
```

modell

nézet

3.Feladat: megvalósítás

```
DirectoryViewer::DirectoryViewer(QWidget *parent) : QDialog(parent)
{
    ...

    _mkdirButton = new QPushButton(tr("&Create Directory..."));
    _removeButton = new QPushButton(tr("&Remove"));
    _quitButton   = new QPushButton(tr("&Quit"));

    connect(_mkdirButton, SIGNAL(clicked()), this, SLOT(createDirectory()));
    connect(_removeButton, SIGNAL(clicked()), this, SLOT(remove()));
    connect(_quitButton,   SIGNAL(clicked()), this, SLOT(accept()));

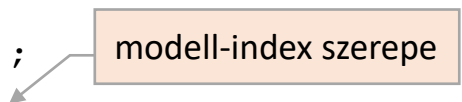
    // elrendezők definiálása
    ...

    setWindowTitle(tr("Directory Viewer"));
}
```

3.Feladat: megvalósítás

```
void DirectoryViewer::createDirectory()
{
    QModelIndex index = _treeView->currentIndex();
    if (!index.isValid()) return;
    QString dirName = QInputDialog::getText(this,
                                             tr("Create Directory"), tr("Directory name"));
    if (!dirName.isEmpty()) {
        if (!model->mkdir(index, dirName).isValid())
            QMessageBox::information(this, tr("Create Directory"),
                                     tr("Failed to create the directory"));
    }
}

void DirectoryViewer::remove()
{
    QModelIndex index = _treeView->currentIndex();
    if (!index.isValid()) return;
    bool ok;
    if (_model->fileInfo(index).isDir()) { ok = model->rmdir(index); }
    else { ok = _model->remove(index); }
    if (!ok) QMessageBox::information(this, tr("Remove"),
                                     tr("Failed to remove %1").arg(_model->fileName(index)));
}
```



modell-index szerepe

Egyedi megjelenítés

- ❑ Lehetőségünk van saját delegált osztályok származtatására is, amelyekben az egyedi megjelenítési módok definiálásához a **paint(...)** metódust kell felülírnunk, mivel ez felel az adatelemek értékének kirajzolásáért.
 - a **drawDisplay** művelettel rajzolhatjuk meg az adatmező felületét
 - a **drawFocus** művelettel pedig erre rárajzolhatjuk a fókusz
 - a **paint** művelet paraméterben megkapja a kirajzoló objektumot (**QPainter**), a kirajzolási stílust (**QStyleOptionViewItem**), valamint a kirajzolandó adatot (**QModelIndex**).
 - A stílusban megfogalmazhatunk különböző módokat (tagolás, igazítás), illetve méretet.
- ❑ A megjelenést táblázat esetén oszloponként is szabályozhatjuk, de hívhatjuk közvetlenül az ősoosztályból örökölt műveletet, így az eredeti viselkedést is visszakaphatjuk.

Példa

```
class MyDelegate : public QItemDelegate
{
...
    void paint(QPainter *painter, ..., const QModelIndex &index) const
    {
        if (index.column() == 2) {      // kettes oszlopra rajzolunk
            ...
            drawDisplay(...);           // adat kirajzolása
            drawFocus(...);             // fókusz kirajzolása
        } else {                        // a többire az alapértelmezettet
            QItemDelegate::paint(...);
        }
    }
...
}
```

4.Feladat

Módosítsuk a korábbi zenei számokat nyilvántartó alkalmazást úgy, hogy a zeneszámok idejének szerkesztését egy számláló elemmel végezhessük.

Track Editor?×

	Track	Duration
1	The Flying Dutchman: Overture	10:30
2	The Flying Dutchman: Wie aus der Fern laengst vergangn	06:14
3	The Flying Dutchman: Steuermann, lass die Wacht	02:32 ▲▼
4	Die Walkuere: Ride of the Valkyries	04:46
5	Tannhaeuser: Freudig begruessen wir die edle Halle	06:24

Add Track

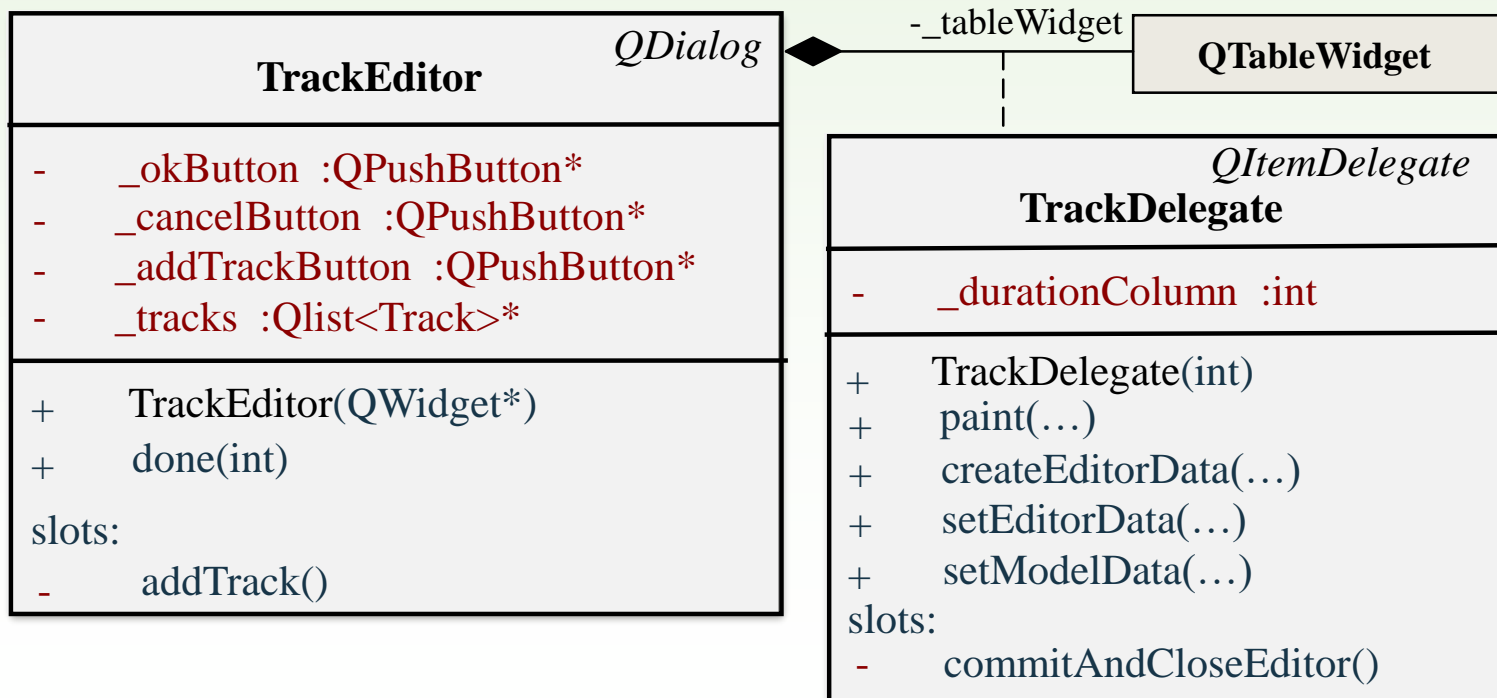
OK

Cancel

4.Feladat: tervezés

- A korábbi megoldáshoz (**TrackEditor**) egy egyedi delegate típust kell definiálni, amely segítségével egy **QTimeEdit** vezérlő fogja az időtartam adatokat megjelenítő **QTableWidgetItem** típusú mezőket módosítani.

```
_tableWidget->setItemDelegate(new TrackDelegate(1));
```



4.Feladat: megvalósítás

```
#include <QApplication>
#include "trackeditor.h"
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QList<Track> tracks;
    tracks << Track("The Flying Dutchman: Overture", 630) <<
        Track("The Flying Dutchman: Wie aus der Fern laengst"
            "vergangner Zeiten", 374)
    << ...
    << Track("Tristan und Isolde: Mild und leise, wie er "
        "laechelt", 375);

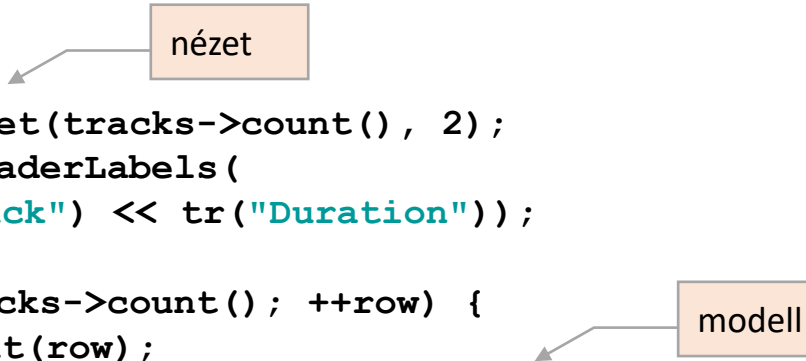
    TrackEditor editor(&tracks);
    editor.resize(600, 300);
    editor.show();

    return app.exec();
}
```

```
class Track
{
public:
    Track(const QString &title = "",
        int duration = 0)
        : _title(title),
          _duration(duration) {}
    QString _title;
    int _duration;
};
```

4.Feladat: megvalósítás

```
TrackEditor::TrackEditor(QList<Track> *tracks, QWidget *parent) :  
    QDialog(parent)  
{  
    _tracks = tracks;  
    tableWidget = new QTableWidgetItem(tracks->count(), 2);  
    tableWidget->setHorizontalHeaderLabels(  
        QStringList() << tr("Track") << tr("Duration"));  
  
    for (int row = 0; row < _tracks->count(); ++row) {  
        Track track = _tracks->at(row);  
        QTableWidgetItem *item0 = new QTableWidgetItem(track.title);  
        tableWidget->setItem(row, 0, item0);  
        QTableWidgetItem *item1 =  
            new QTableWidgetItem(QString::number(track.duration));  
        item1->setTextAlignment(Qt::AlignRight);  
        tableWidget->setItem(row, 1, item1);  
    }  
    tableWidget->resizeColumnToContents(0);  
    ...  
}
```



nézet

modell

4.Feladat: megvalósítás

```
TrackEditor::TrackEditor(QList<Track> *tracks, QWidget *parent) :
    QDialog(parent)
{
    ...
    _addTrackButton = new QPushButton(tr("&Add Track"));
    _okButton = new QPushButton(tr("OK"));
    _okButton->setDefault(true);
    _cancelButton = new QPushButton(tr("Cancel"));

    connect(_addTrackButton, SIGNAL(clicked()), this, SLOT(addTrack()));
    connect(_okButton,      SIGNAL(clicked()), this, SLOT(accept()));
    connect(_cancelButton,  SIGNAL(clicked()), this, SLOT(reject()));

    // elrendezők definiálása
    ...

    setWindowTitle(tr("Track Editor"));
}
```

4.Feladat: megvalósítás

```
TrackDelegate::TrackDelegate(int durationColumn, QObject *parent) :
QItemDelegate(parent)
{
    this->_durationColumn = durationColumn;
}

void TrackDelegate::paint(QPainter *painter, const QStyleOptionViewItem
&option, const QModelIndex &index) const
{
    if (index.column() == _durationColumn) {
        int secs = index.model()->data(index, Qt::DisplayRole).toInt();
        QString text = QString("%1:%2").
            arg(secs/60, 2, 10, QChar('0')).arg(secs%60, 2, 10, QChar('0'));

        QStyleOptionViewItem myOption = option;
        myOption.displayAlignment = Qt::AlignRight | Qt::AlignVCenter;
        drawDisplay(painter, myOption, myOption.rect, text);
        drawFocus(painter, myOption, myOption.rect);
    } else { QItemDelegate::paint(painter, option, index); }
}
```

egy adatelem kirajzolásakor fut le

modell-index szerepe

4.Feladat: megvalósítás

egy adatelem szerkesztésének kezdetekor :

1. létrejön a szerkesztés vezérlője (createEditor)
2. amelybe aztán megfelelő tartalom töltődik be (setEditorData)

```
QWidget *TrackDelegate::createEditor(QWidget *parent,
    const QStyleOptionViewItem &option, const QModelIndex &index) const
{
    if (index.column() == _durationColumn) {
        QTimeEdit *timeEdit = new QTimeEdit(parent);
        timeEdit->setDisplayFormat("mm:ss");
        connect(timeEdit, SIGNAL(editingFinished()), this,
            SLOT(commitAndCloseEditor()));
        return timeEdit;
    } else { return QItemDelegate::createEditor(parent, option, index); }
}

void TrackDelegate::setEditorData(QWidget *editor,
    const QModelIndex &index) const
{
    if (index.column() == _durationColumn) {
        int secs = index.model()->data(index, Qt::DisplayRole).toInt();
        QTimeEdit *timeEdit = qobject_cast<QTimeEdit *>(editor);
        timeEdit->setTime(QTime(0, secs / 60, secs % 60));
    } else { QItemDelegate::setEditorData(editor, index); }
}
```

4.Feladat: megvalósítás

egy adatelem szerkesztésének befejezésekor (enter, fókusz, ...) :

1. értesíteni kell a szerkesztést végző vezérlőt (commitAndCloseEditor)
2. frissíteni kell a modellt (setModelData)

```
void TrackDelegate::commitAndCloseEditor()
{
    QTimeEdit *editor = qobject_cast<QTimeEdit *>(sender());
    emit commitData(editor);
    emit closeEditor(editor);
}

void TrackDelegate::setModelData(QWidget *editor,
    QAbstractItemModel *model, const QModelIndex &index) const
{
    if (index.column() == _durationColumn) {
        QTimeEdit *timeEdit = qobject_cast<QTimeEdit *>(editor);
        QTime time = timeEdit->time();
        int secs = (time.minute() * 60) + time.second();
        model->setData(index, secs);
    } else { QItemDelegate::setModelData(editor, model, index); }
}
```

Adatbáziskezelés

Adatbázisok használata Qt alatt

- ❑ A Qt-ben a *QtSql* module támogatja az SQL elérésű adatbázisok platform- és adatbázis független használatát.
- ❑ Az adatbázis kapcsolat kiépítése során megadott 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 táblákban tudják betölteni, majd ezekben, mint adatmodellekben, az adatokat bejárhatják és szerkeszthetik, majd az adatbázisba visszaírhatják. Ezek a modellek megfelelő 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
 - QSQLITE ~ SQLite
- ❑ Nincs minden meghajtó előre telepítve, bizonyos esetekben további csomagként (pl. `libqt5sql5-mysql`) kell hozzáadnunk őket.

A QSql modul használatba vétele

- ❑ A modul osztályainak használatához a megfelelő könyvtárt kell meghivatkozni az aktuális fájlban, illetve lehetőség van a teljes modul betöltésére is:

#include <QSql>

- ❑ A modul alapból nem érhető el egy alap Qt alkalmazásban, használatát a projektfájlban jeleznünk kell a **QT += sql** utasítással

- a betöltendő modulok egy sorban is lehetnek, pl.

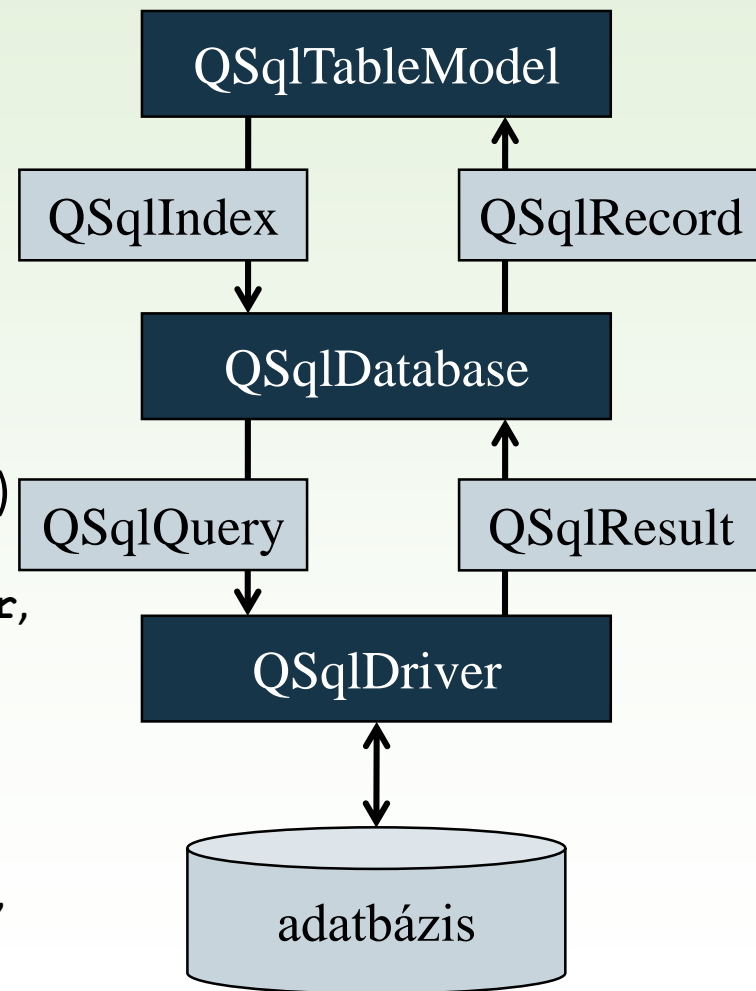
QT += core gui widgets sql

- amennyiben konzol felületen szerkesztünk, a projektfájl létrehozásakor is hozzáadhatjuk a modult:

qmake -project "QT += sql"

A Qt adatbázis-kezelő modul

- A *QtSql* modul számos osztályt tartalmaz az SQL alapú adatbázisok kezeléséhez. Ezeket három csoportba sorolhatjuk.
 - A **modell osztályok** biztosítják a logikai (memóriában felépített) adatbázist, és ehhez interfészt a felületi réteg számára: `QSqlQueryModel`, `QSqlTableModel`, `QSqlRelationalModel`.
 - Az **alkalmazásprogramozói osztályok** (API) biztosítják az SQL elemek kezelését: `QSqlDataBase`, `QSqlQuery`, `QSqlError`, `QSqlResult`, `QSqlField`, `QSqlIndex`, `QSqlRecord`.
 - A **meghajtó osztályok** biztosítják az adatbázis elérését és a kommunikációt: `QSqlDriver`, `QSqlDriverCreator<T>`, `QSqlDriverCreatorBase`, `QSqlDriverPlugin`.



Adatbáziskapcsolat létesítése

- ❑ Adatbázismotort betölteni, és kapcsolatot létesíteni a **QSqlDatabase** osztály segítségével tudunk, pl.:

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

- ❑ A betöltést az **addDatabase()** statikus metódussal végezzük a meghajtó megadásával, beállíthatjuk a szerveret, az adatbázist, valamint a felhasználó adatait.
- ❑ A rendelkezésre álló meghajtókat a **drivers()** metódussal kérhetjük le.

Adatbáziskapcsolatok

- ❑ Lehetőségünk van több kapcsolatot is kezelni a programban:
 - A kapcsolatok elnevezéssel rendelkeznek, ezen keresztül érhetjük el őket: **addDatabase (<meghajtó>, <név>)** .
 - Kapcsolatok listázhatóak (**connectionNames()**), lekérhetőek (**database (<név>)**), törölhetőek (**removeDatabase (<név>)**).
- ❑ Kapcsolatot megnyitni az **open()**, bezárni a **close()** művelettel tudunk.
 - 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();
```

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

Lekérdezés olvasása

- ❑ Amennyiben lekérdezést hajtottunk végre, az eredményt soronként kezeljük, azaz egyszerre nem látjuk a teljes eredményt csak egy sorát.
 - Lépegetni a **first()**, **next()**, **previous()**, **last()** utasításokkal lehet, a **size()** megadja a lekérdezett sorok számát.
 - Kezdetben az eredmény első sora előtt állunk, tögtén léptetni kell.
 - Amennyiben csak előre akarunk lépkedni, a **setForwardOnly()** metódus optimalizálja a lekérdezést.
 - Adott sorra ugrani a **seek(<sorszám>)** metódussal, mindegyik igazat ad, ha tudott lépni.
 - Értéket lekérdezni a kijelölt sorból a **value(<oszlopszám>)** metódussal tudunk. Az érték **QVariant** típusú, így az tovább konvertálható alkalmas formára.

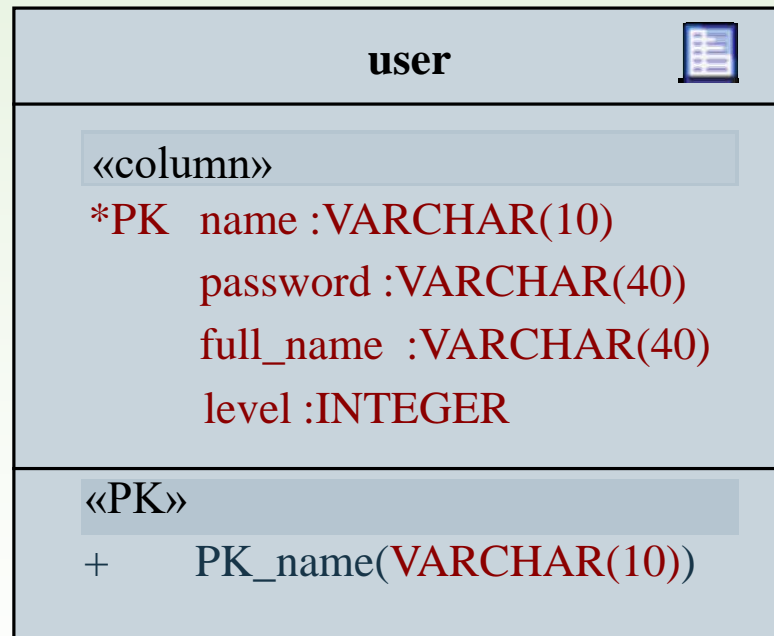
```
while (query.next()) {  
    cout << query.value(0).toString(); // első oszlop szöveg  
    cout << query.value(1).toInt();   // második oszlop egész szám  
}
```

1.Feladat

Készítsünk egyszerű konzol alkalmazást, amely alkalmas az apartman-adatbázis (**apartments**) felhasználók (**user**) táblájának listázására, felhasználó törlésére, valamint új felhasználó beszúrására.

- A program csak megfelelő felhasználónév/jelszó megadásával engedi elvégezni a tevékenységeket.
- A program lekérdezés segítségével azonosít, beolvassa a táblatartalmat (azonosító, név, jelszó, szint), kilistázza, lehetőséget ad azonosító alapján törlésre, és beszúrásra.
- A programot vezéreljük menün keresztül (ehhez hozzunk létre egy menü osztályt).

1.Feladat: tervezés



1.Feladat: tervezés

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

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

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

```
insert into user(name, password, full_name, level)  
    values('root', 'root', 'admin', 0);  
insert into user(name, password, full_name, level)  
    values('gt', 'secret', 'Gregorics', 1);
```

1.Feladat: adatbázis kapcsolat

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

sikeres megnyitás esetén

kapcsolat bezárása

sikertelen kapcsolódás

1.Feladat: Menu osztály

- ❑ Hozzunk létre egy menü osztályt egy menü kezelésére. Legyen benne négy menüpont, plusz a kilépés.

Menu	
+	Menu()
+	run() :void
-	showAllUsers() :void
-	showCreateUser() :void
-	showRemoveUser() :void
-	validateUser() :bool

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().toString()
        << setw(20) << selectQuery.value(1).toString().toString()
        << setw(20) << selectQuery.value(2).toString().toString()
        << setw(20) << selectQuery.value(3).toInt() << endl;
    }
}
```

1.Feladat: beszúrás

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

← adatok beolvasása

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) + "'");  
}
```

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

ha van eredmény akkor
sikerkült 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 szöveggént állítanak össze.
 - Az összeállítás során törekedni kell arra, hogy ne lehessen manipulálni az utasítást úgy, hogy az illetéktelen felhasználók hozzáférhessenek vagy kárt tegyenek a tárolt adatokhoz/ban.
- ❑ Az SQL parancsok manipulációját nevezzük *SQL injekciónak* (*SQL injection*)
 - leggyakoribb webes környezetben

Példa

```
// felhasználónév/jelszó bekérése

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

Ha a felhasználó az alábbi adatokat adja meg:

name = ""

password = "' or '1' = '1"

akkor a következő lekérdezés valósul meg:

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

amelynek feltétele a tábla minden sorára igaz, tehát hozzájutunk a tábla összes nevéhez.

SQL injekció kivédése

1. A felhazsnálótól kapott értékek ellenőrzésével, módosításával.

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

2. A felhasználó által megadható adatok korlátozásával.

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

3. Paraméteres utasítások használatával (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();
```

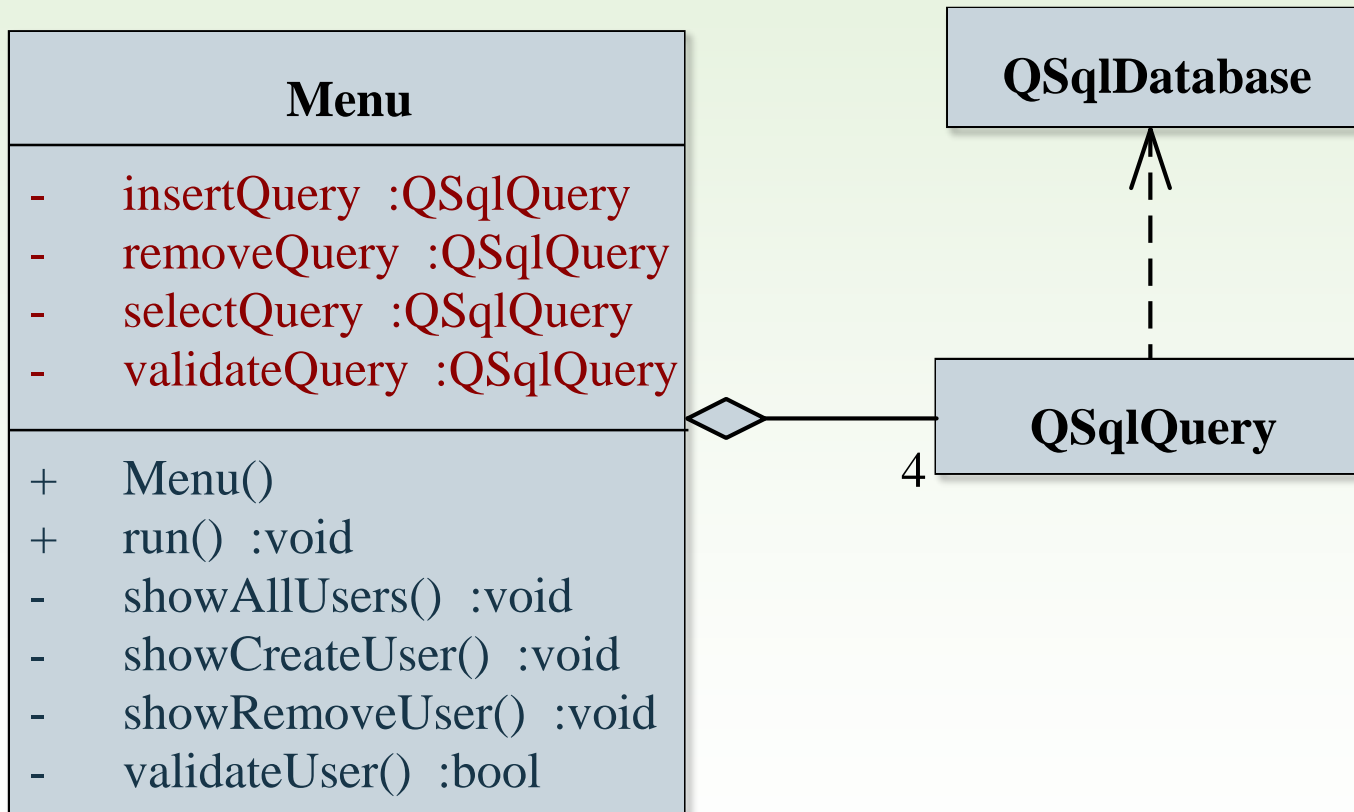
4. Az injekciót eleve kizáró megoldások (pl. adatmodell) használatával.

2.Feladat

Készítsünk egyszerű konzol alkalmazást, amely alkalmas az apartman-adatbázis (**apartments**) felhasználók (**user**) táblájának listázására, felhasználó törlésére, valamint új felhasználó beszúrására.

- A program csak megfelelő felhasználónév/jelszó megadásával engedi elvégezni a tevékenységeket.
- A program paraméteres utasításokat fog használni, kikerülve az SQL injekció lehetőségét.
- Azonosít, beolvassa a táblatartalmat (azonosító, név, jelszó, szint), kilistázza, lehetőséget ad azonosító alapján törlésre, és beszúrásra.

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

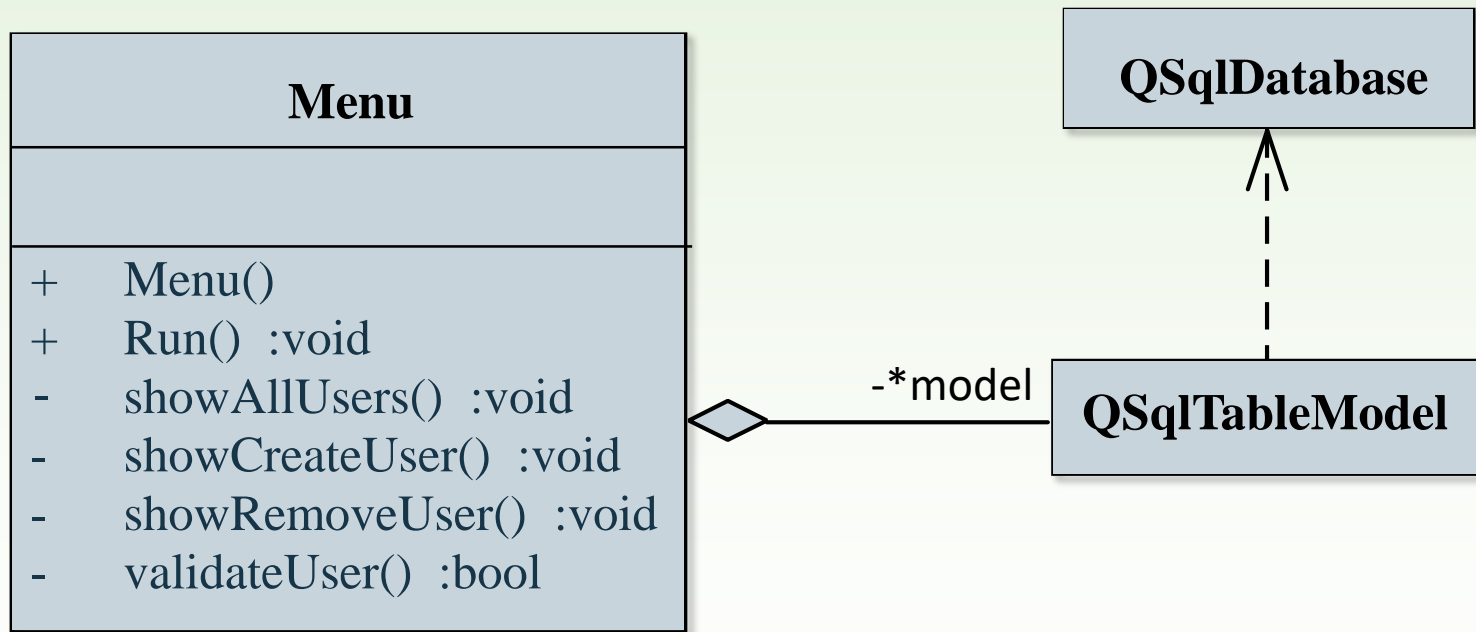
ha van eredmény akkor
sikerkült a lekérdezés

3.Feladat

Készítsünk egyszerű konzol alkalmazást, amely alkalmas az apartman-adatbázis (**apartments**) felhasználók (**user**) táblájának listázására, felhasználó törlésére, valamint új felhasználó beszúrására.

- Az alkalmazáshoz csak a korábbi menü osztályt kell módosítanunk, további saját új osztályra nem lesz szükségünk, a létező típusok felhasználásával megoldható a feladat.
- Az adatok kezelését egy tábla modellel 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");  
    // lehetne szűrőt adni: model->setFilter("level = 0")  
}
```

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